CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Victor Machado Emerick

IMPLEMENTAÇÃO DE UM AMBIENTE SIMPLIFICADO PARA A ARQUITETURA MIPS MONOCICLO EM PLACA DE DESENVOLVIMENTO FPGA

Timóteo

2019

Victor Machado Emerick

IMPLEMENTAÇÃO DE UM AMBIENTE SIMPLIFICADO PARA A ARQUITETURA MIPS MONOCICLO EM PLACA DE DESENVOLVIMENTO FPGA

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Elder de Oliveira Rodrigues

Timóteo

2019

Ama-se mais o que se conquista com esforço.
Benjamin Disraeli

Agradecimentos

Sou grato, primeiramente, a Deus por toda sua misericórdia e amor durante todo o meu processo de formação no Curso de Engenharia de Computação.

Agradeço à minha família o carinho, apoio e dedicação à minha educação, desde a infância.

Sou grato a Ana Clara de Assis Alves pelo companheirismo, carinho, sugestões e auxílio na escrita deste trabalho.

Também agradeço aos meus amigos o incentivo na escolha e prosseguimento ao tema escolhido e ao auxílio na solução de problemas relacionados.

Agradeço ao professor Elder de Oliveira Rodrigues os ensinamentos, a paciência, a clareza e as orientações em conteúdos, esclarecimentos de dúvidas e apoio fornecidos, não somente durante o curso, mas também durante a produção deste trabalho.

E,finalmente, agradeço aos meus professores que compartilharam conosco seus conhecimentos, para que pudéssemos chegar ao final do curso com excelência e confiança de sermos pessoas e profissionais melhores.

“Os grandes navegadores
devem sua reputação aos temporais e tempestades”.
Epicuro

Resumo

Este trabalho é caracterizado pela implementação de um ambiente de desenvolvimento, que visa dar maior compreensão à forma de interação e visualização do fluxo de dados da arquitetura MIPS (Microprocessor without Interlocked Pipeline Stages) monociclo do livro “Organização e Projeto de Computadores: Interface Hardware/Software” através de um dispositivo lógico programável em comunicação com um monitor e um teclado. Verifica-se, na literatura sobre o tema, que estes trabalhos, quando envolvem a tecnologia de FPGA (Field Programmable Gate Array), apresentam limitação visual e interativa. Neste contexto, é proposto um ambiente que visa melhor auxiliar o usuário a compreender o comportamento desta arquitetura. Importante salientar que todos os programas descritos em VHDL, monociclo, VGA, teclado, caminho de dados da arquitetura, exibido como imagem a cores e seus valores dinâmicos, e a própria imagem da arquitetura, bem como todos os processos de interligação, que se fazem necessários, como comunicação entre as partes, estão sintetizados em um único dispositivo FPGA. Ressalta-se que não há necessidade de re-embarcar um novo código para ser executado, uma vez que o programa digitado passa por processo de Montagem, conforme instruções já mapeadas por máquina de estado, transformado em hardware. Como resultado, obteve-se um ambiente inteiramente descrito em hardware com capacidade para editar códigos, armazenar dados e caracteres, executar instruções básicas, exibir o caminho de dados do MIPS, bem como seus valores, que se encontram na comunicação entre os componentes da arquitetura e a criação de novos programas para execução. Testes feitos com o Kit Altera-DE2, monitor e teclado com programas de instruções básicas (add, sub, and, or, slt, lw, sw, beq, j e addi) foram realizados, obtendo sucesso na execução e na exibição das imagens.

Palavras-chave: VHDL, MIPS, FPGA, VGA, PS/2, teclado, monitor.

Abstract

The main propose of this work is the implementation of a development environment, that improve way of visualization and interaction of monocycle architecture of MIPS (Microprocessor without Interlocked Pipeline Stages) data path of the book "Computer Organization and Design: The Hardware/Software Interface" with a programmable logic device interfacing a monitor and keyboard. In literature that these works when involve FPGA (Field Programmable Gate Array) technology are visually and interactively restricted. In this context, a environment is proposed that aims to help the user to understand the behavior of the architecture. Importantly, all the programs described in VHDL like monocycle, VGA and keyboard controllers, architecture data path displayed as a color image and its dyanmic values and its own imagem of architecture, as well as all necessary interconnection processes as communication between the parts are synthesized in a single FPGA device. It is noteworthy that there is no need to re-embark a new code to be executed, since the program entered undergoes assembly process according to the instructions already mapped by state machine has transformed into hardware. As a result, a fully described hardware environment was obtained with the ability to: edit codes, store data and characters, perform basic instructions, display the MIPS data path, and their values found in the communication between the components of the architecture and the creation of new programs for execution. Tests performed with the Altera-DE2 Kit, monitor and keyboard with basic instruction programs (add, sub, and, or, sl, lw, sw, beq, j and addi) were performed successfully in the execution and display of images.

Keywords: VHDL, MIPS, FPGA, VGA, PS/2, keyboard, monitor.

Lista de ilustrações

Figura 1 – Janela de execução do MARS ativa (A aba "Execute"está em primeiro plano e a barra de ferramentas de execução está ativa). 16
Figura 2 – Tela de execução do ViSiMIPS exibindo o caminho de dados da arquitetura MIPS. 17
Figura 3 – Uma visão abstrata de um FPGA; células lógicas são embarcadas em uma estrutura geral de roteamento. 21
Figura 4 – Um autômato finito que possui três estados. 23
Figura 5 – Um autômato finito não-determinístico. 23
Figura 6 – Computação determinística e não-determinística com ramos de aceitação. . 24
Figura 7 – Conector VGA com identificação dos pinos e cores representativas. 25
Figura 8 – Características temporais dos sinais de sincronização. 26
Figura 9 – Pinagem de cada conector. (Adaptado) 27
Figura 10 – Caminho de dados e controle simples para tratamento de instruções de salto. 29
Figura 11 – Divisão de campos de instruções do tipo R nomeados 30
Figura 12 – Bloco de memória SDRAM, da Hynix® HY57V641620FTP, apresentando e agrupando os pinos presentes no chip. 34
Figura 13 – Formas de onda demonstrando a temporização de execução dos comandos *Bank Active Read* e *Write*. 36
Figura 14 – Passos no processo de compilação 38
Figura 15 – Formatos de instruções: A) *register-to-register*, B) memória. 39
Figura 16 – Janela de simulação. 40
Figura 17 – Fluxo para exportação da implementação para prototipação em FPGA. . . . 41
Figura 18 – Notebook utilizado durante o desenvolvimento da ferramenta. 42
Figura 19 – Placa de desenvolvimento EasyFPGA v2.2a utilizada na produção desta pesquisa. 44
Figura 20 – Placa de desenvolvimento Altera DE2-35C utilizada na produção desta pesquisa. 45
Figura 21 – Fluxograma das etapas deste trabalho. 47
Figura 22 – Arquitetura proposta para o desenvolvimento do ambiente em FPGA. 49
Figura 23 – Arquitetura proposta para o desenvolvimento do ambiente em FPGA utilizando memória *on-chip*. 50
Figura 24 – Caminho de dados da arquitetura MIPS. 51
Figura 25 – Caminho de dados da arquitetura MIPS desenhado no software Hades. . . . 53
Figura 26 – Simulação da arquitetura MIPS em formas onda executando a instrução *sub t1 t2 t3*. 55
Figura 27 – Estrutura de um simples controlador VGA no modo de texto. 56
Figura 28 – Estrutura da controladora gráfica. 57
Figura 29 – Tela principal do software BlockPaint. 61
Figura 30 – Tela de desenho de fontes do software BlockPaint. 62

Figura 31 – Exemplo de como é gerado o código de armazenamento em memória. . . . 63
Figura 32 – Exemplo de como é gerado o código de blocos de 'if's. 64
Figura 33 – Tela em que são gerados os códigos de configuração do controlador. 65
Figura 34 – Tela de teste exibindo os caracteres desenhados no software. 66
Figura 35 – Exemplo de detecção de borda de *clock*. 67
Figura 36 – Autômato finito definido para a montagem do código de acordo com as instruções definidas. 68
Figura 37 – Divisão interna do bloco de arquitetura e suas entradas e saídas. 71
Figura 38 – Prévia da imagem a ser exibida da arquitetura produzida pelo software BlockPaint. 74
Figura 39 – Exibição da tela da ferramenta. 77
Figura 40 – Ferramenta finalizada. 78
Figura 41 – Exibição do programa gravado na memória ROM, dados na memória RAM e banco de registradores. 79
Figura 42 – Exibição da execução da instrução ADDI. 82
Figura 43 – Exibição da execução da instrução BEQ. 83

Lista de tabelas

Tabela 1 – Lista de pinos com numeração referente à figura 7 25
Tabela 2 – Temporização de diferentes resoluções. 26
Tabela 3 – Corpo da mensagem durante a comunicação. (Adaptado) 27
Tabela 4 – Tabela de tradução dos *scan codes* de teclados. (Adaptado) 28
Tabela 5 – Formatos de instruções do MIPS. 31
Tabela 6 – Formato do comando *Mode Register Set* da memória Hynix® HY57V641620FTP. (Adaptado) 35
Tabela 7 – Combinação de sinais necessários para executar o comando *Bank Active* na memória. (Adaptado) 36
Tabela 8 – Combinação de sinais necessários para executar o comando *Bank Active* na memória. (Adaptado) 36
Tabela 9 – Instruções com suas formas de uso e funcionamento. 52
Tabela 10 – Parâmetros detalhados de temporização para a resolução 1360x768 (Adaptado). 59
Tabela 11 – AC CHARACTERISTICS II. 69
Tabela 12 – Relatório de compilação do uso de recursos do Quartus. 76
Tabela 13 – Relatório de utilização de recursos após otimização. 77

Lista de abreviaturas e siglas

AF	Autômato Finito
AFD	Autômato Finito Determinístico
AFN	Autômato Finito Não-determinístico
ALU	Arithmetic Logic - Unidade Lógico-Aritmética
ASCII	American Standard Code for Information Interchange - Código Padrão Norte-americado para Intercâmbio de Informações
ASIC	Application Specific Integrated Circuits - Circuitos Integrados de Aplicação Específica
BA	Bank Address - Endereço do Banco - ou Bloco de Arquitetura
BL	Burst Length - Comprimento do Disparo
BP	Back Porch - Porção Traseira
CA	Column Address - Endereço da Coluna
CG	Controladora Gráfica
CLPD	Complex Programmable Logic Device - Dispositivo Complexo Lógico Programável
CM	Controlador de Memória
CRT	Catode-Ray Tube - Tubo de Raios Catódicos
DRAM	Dynamic Random Access Memory - Memória Dinâmica de Acesso Randômico
ET	Editor de Texto
FP	Front Porch - Porção Frontal
FPGA	Field Programmable Gate Array - Campo de Arranjo de Portas Programáveis
GM	Gerenciador de Memória
HBP	Horizontal Back Porch - Porção Horizontal Traseira
HDL	Hardware Description Language - Linguagem de Descrição de Hardware
HDMI	High-Definition Multimedia Interface - Interface Multimídia de Alta Definição
HFP	Horizontal Front Porch - Porção Horizontal Frontal

HSP	Horizontal Sync Pulse - Pulso de Sincronia Horizontal
IBM	Internation Business Machines - Empresa Internacional de Máquinas
ISA	Industry Standard Architecture - Arquitetura Padrão da Indústria
LCD	Liquid Crystal Display - Visor de Cristal Líquido
LE	Logic Element - Elemento Lógico
LUT	LookUp Table - Tabela de Pesquisa
MARS	MIPS Assembler and Runtime Simulator - Montador e Simulador de Tempo de Execução do MIPS
MIPS	Microprocessor without Interlocked Pipeline Stages - Microprocessador sem Estágios Intertravados de Pipeline
PC	Pixel Clock - Clock de Pixel - ou Processador de Comandos ou Program Counter - Contador de Programa
PC	Processador de Comandos
PLL	Phased-Lock Loop - Malha de Captura de Fase
PS/2	Personal System/2 - Sistema Pessoal/2
RA	Row Address - Endereço da Linha
RAM	Random Access Memory - Memória de Acesso Randômico
ROM	Read-Only Memory - Memória apenas para Leitura
RTL	Register Transfer Level - Nível de Transferência de Registradores
SDRAM	Synchornous Dynamic Random Access Memory - Memória Síncrona Dinâmica de Acesso Randômico
SP	Sync Pulse - Pulso de Sincronia
SPLD	Simple Programmable Logic Device - Dispositivo Simples Lógico Programável
TR	Tradutor
ULA	Unidade Lógica e Aritmética
USB	Universal Serial Bus - Barramento Serializado Universal
VBP	Vertical Back Porch
VESA	Video Eletronics Standards Association - Associação de Padrões de Vídeo de Eletrônicos

VFP	Vertical Front Porch
VGA	Video Grid Array - Padrão de Disposição Gráfica
VHDL	VHSIC Hardware Description Language - Linguagem de Descrição VHSIC
VHSIC	Very High-Speed Integrated Circuit - Circuitos Integrados de Alto Desempenho
ViSiMIPS	Visual Simulation MIPS - Simulação Visual do MIPS
VSP	Vertical Sync Pulse - Pulso de Sincronia Vertical
WYSIWYG	What You See Is What You Get - O que Você Vê é o que Você Tem

Sumário

1 INTRODUÇÃO 15
1.1 Problema 17
1.2 Justificativa 18
1.3 Objetivos 18
1.4 Estrutura do Trabalho 19

2 REVISÃO BIBLIOGRÁFICA 20
2.1 Fundamentos Teóricos 20
2.1.1 FPGA 20
2.1.2 Autômatos Finitos 22
2.1.3 VGA 24
2.1.4 PS/2 27
2.1.5 Arquitetura Monociclo MIPS 29
2.1.6 *Synchronous Dynamic Random Access Memory* (SDRAM) 32
2.1.6.1 Inicialização 34
2.1.6.2 Leitura e Gravação 35
2.1.7 Montador (*Assembler*) 37
2.2 Fundamentos Históricos 38
2.2.1 Um Ambiente Gráfico de Simulação Baseado no MIPS em VHDL 38
2.2.2 MIPSFPGA - Um Simulador MIPS Incremental com Validação em FPGA . . . 39

3 MATERIAIS E MÉTODOS 42
3.1 Materiais 42
3.2 Classificação 46
3.3 Procedimentos Metodológicos 46

4 CONSTRUÇÃO DO AMBIENTE E PROGRAMAÇÃO DA FERRAMENTA PARA A ARQUITETURA MIPS MONOCICLO 48
4.1 Arquitetura MIPS Monociclo 50
4.2 Estudo de VHDL e Implementação da Arquitetura MIPS em VHDL 54
4.3 Controladores 54
4.3.1 Controladora Gráfica 54
4.3.1.1 Software de desenho em blocos 59
4.3.1.1.1 Janela Principal 60
4.3.1.1.2 Desenho de Caracteres 62
4.3.1.1.3 Geração de Código VHDL 62
4.3.1.1.4 Geração de código de configuração do controlador. 64
4.3.1.1.5 Finalização 65
4.3.2 Controlador PS/2 66

4.3.3 Montador 68
4.4 Controladora de Memória SDRAM 69
4.5 Blocos de Memória RAM 70
4.6 Bloco de Arquitetura 71
4.6.1 Processador de Imagem 72
4.7 Processador de Comandos e Editor de Texto 72

5 RESULTADOS 75
5.1 Análise de dados de compilação. 75
5.2 Análise da interface gráfica e interação 77
5.3 Analise de Execução de Instruções 78

6 CONCLUSÃO 84
6.1 Trabalhos Futuros 85

REFERÊNCIAS 87

1 Introdução

“A vida não examinada não vale a pena ser vivida”.
Aristóteles

Os simuladores são softwares que auxiliam na compreensão do funcionamento de algum sistema. Isso é aplicável, por exemplo, ao ensinar um aluno a compreender o comportamento da arquitetura MIPS (*Microprocessor without Interlocked Pipeline Stages*). De acordo com Kabir, Bari e Haque (2011), baseado no conhecimento adquirido ao usar simuladores, os alunos são motivados a estudar além dos objetivos pressupostos nos livros.

O MIPS, exemplo de arquitetura de computador, foi desenvolvido por Hennessy e Patterson em 1981, com o intuito de aumentar o desempenho da execução de instruções utilizando estágios de *pipelines*. Segundo Patterson e Hennessy (2012, p.346, tradução nossa), o pipeline possui o papel de aumentar o número de instruções executadas simultaneamente, além de elevar a taxa com as quais essas são iniciadas e finalizadas [1]. A otimização descrita é feita por meio da divisão da execução de instruções em etapas, permitindo executar várias instruções de maneira sobreposta.

Existem diversos simuladores da arquitetura MIPS que possuem diferentes funções, tais como permitir a visualização do carregamento e da coleta de dados da memória RAM (*Random Access Memory*) e do banco de registradores, executar os códigos da linguagem *Assembly* MIPS e, também, possibilitar o acompanhamento da execução do programa passo-a-passo. Ademais, esses simuladores devem possuir um montador, ou *assembler*, capaz de interpretar todas as instruções que a arquitetura propõe, visto que, para se ter uma experiência completa, é necessário que toda a arquitetura esteja implementada.

Dentre os simuladores existentes para a arquitetura MIPS, pode-se citar:

- MARS (*MIPS Assembler and Runtime Simulator*), desenvolvida por Vollmar e Sanderson (2006).
- ViSiMIPS (*Visual Simulator MIPS*), desenvolvido por Kabir, Bari e Haque (2011);

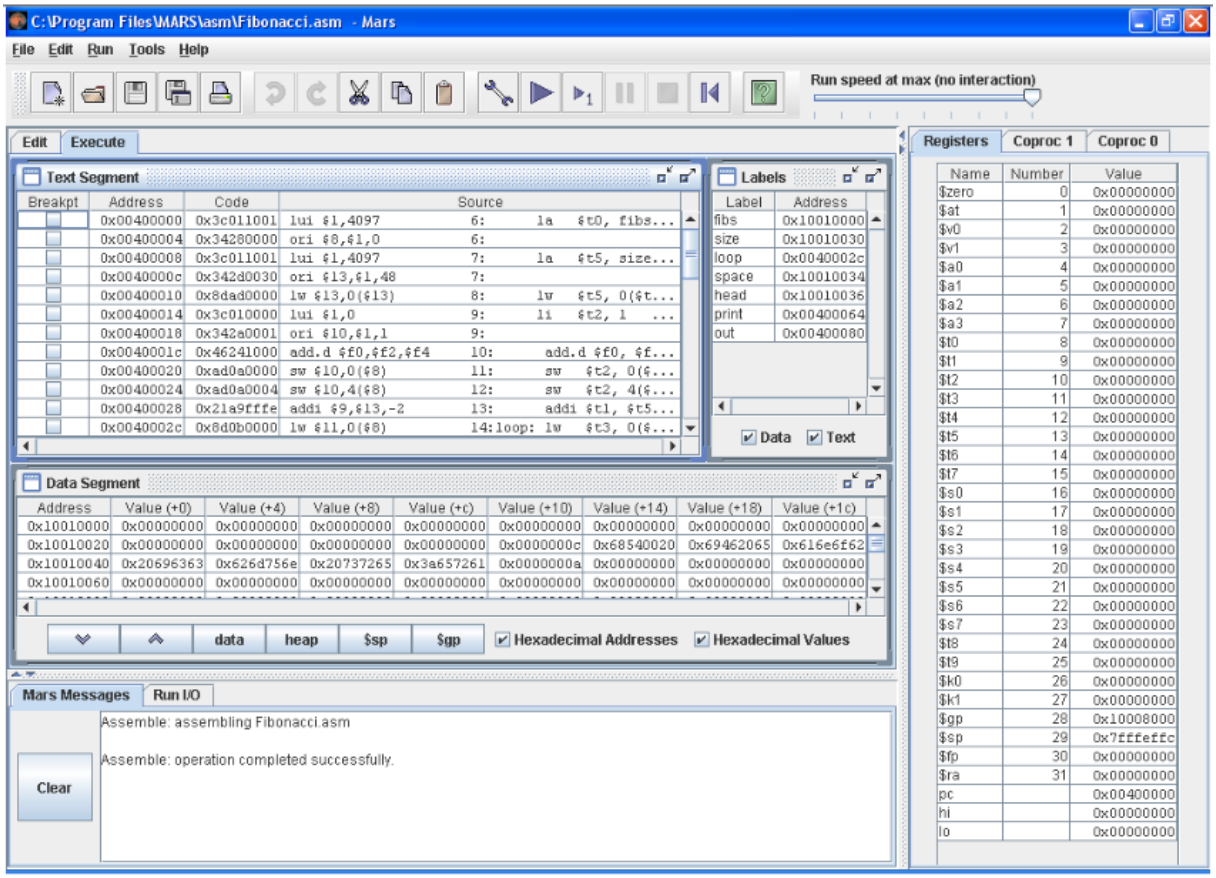
O primeiro, MARS, trata-se de um simulador educacional que foi criado para ser uma alternativa ao SPIM (Anagrama de MIPS escrito ao contrário), que é um simulador apresentado no livro de Hennessy e Patterson (2008), Organização e Projeto de Computadores.

> Nosso objetivo para este projeto foi criar uma alternativa ao SPIM, especificamente para as típicas necessidades de estudantes de graduação e seus instrutores. Ele pode ser utilizado em cursos como arquitetura e organização de

[1] *Pipelining increases the number of simultaneously executing instructions and the rate at which instructions are started and completed.* (PATTERSON; HENNESSY, 2012)

> computadores, programação em linguagem *assembly* e escrita de compilador. (VOLLMAR; SANDERSON, 2006, tradução nossa, p.1). [2]

Figura 1 – Janela de execução do MARS ativa (A aba "Execute"está em primeiro plano e a barra de ferramentas de execução está ativa).



(VOLLMAR; SANDERSON, 2006)

Na figura 1, é possível observar a interface gráfica utilizada pelo MARS durante a execução de um determinado programa desenvolvido na própria ferramenta, demonstrando o passo-a-passo de acordo com a execução das instruções, os dados armazenados nos registradores, na memória e, ainda, uma área de interação com usuário para execução de *syscalls*.

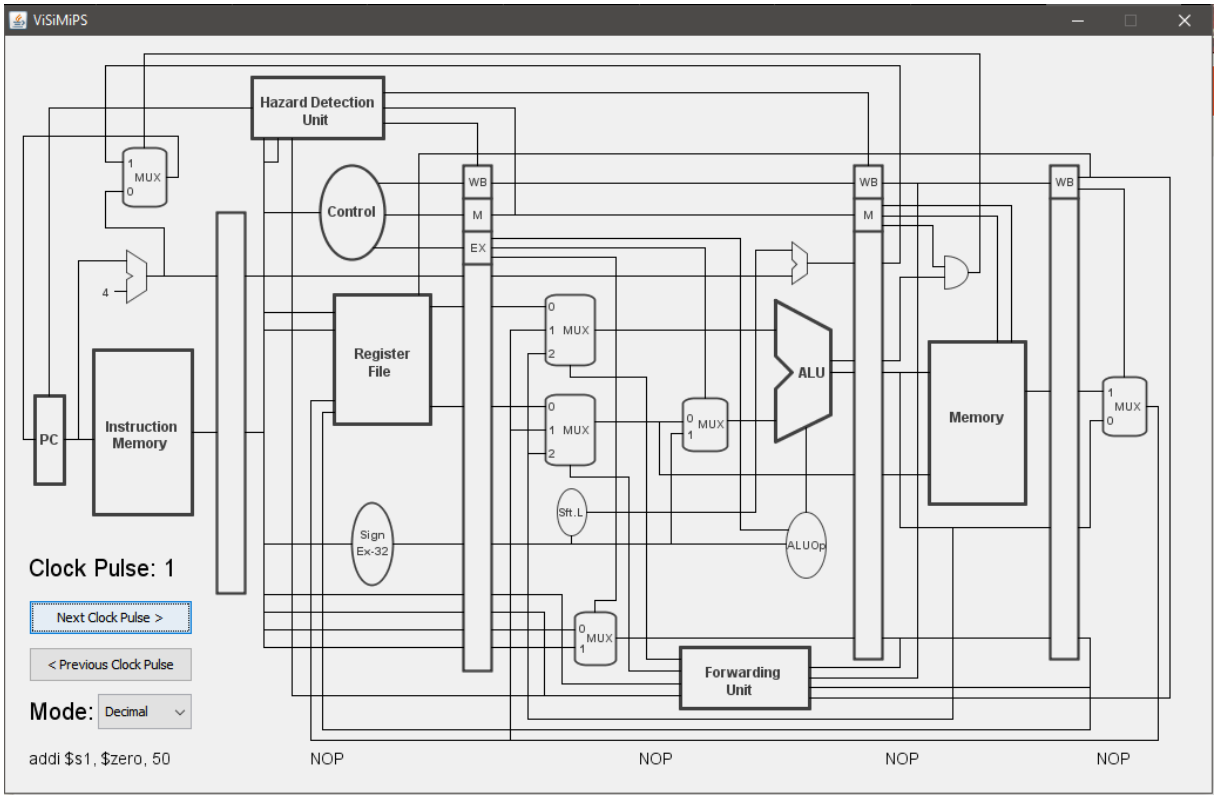
O ViSiMIPS, por sua vez, é um simulador da arquitetura MIPS com cinco estágios de *pipeline* e permite visualizar o fluxo de dados. Assim como o MARS, e segundo Kabir, Bari e Haque (2011), o ViSiMIPS foi desenvolvido com a intenção de contribuir com o ensino e fornecer aos estudantes um ambiente para analisar o funcionamento do MIPS.

Na figura 2, é ilustrado o ViSiMIPS em seu estado de execução. No geral, o simulador é simples e atende ao seu principal propósito: exibir, graficamente, a execução das instruções no caminho de dados. A codificação é realizada em uma caixa de texto que é exibida ao iniciar o *software* e a execução é sempre passo-a-passo, simulando pulsos de *clock*. Além disso, o

[2] Our goal for this project was to create an alternative to SPIM specifically for the needs of typical undergraduate students and their instructors. It should be useful in courses such as computer organization and architecture, assembly language programming, and compiler writing. (VOLLMAR; SANDERSON, 2006).

programa permite visualizar os dados que percorrem a arquitetura, seja apontando o mouse nas linhas que conectam os componentes ou clicando na área do banco de registradores. No entanto, essa funcionalidade não se aplica ao bloco de memória RAM e de instruções.

Figura 2 – Tela de execução do ViSiMIPS exibindo o caminho de dados da arquitetura MIPS.



(KABIR; BARI; HAQUE, 2011)

No presente trabalho, um simulador é desenvolvido e embarcado em um chip FPGA (*Field Programmable Gate Array*). O chip FPGA corresponde a um circuito integrado que é configurado, usando uma linguagem HDL (*Hardware Description Language*, em inglês, Linguagem de Descrição de Hardware) para ter o comportamento de um hardware qualquer dentro de suas limitações. Com isso, um hardware, descrito e prototipado em FPGA, será um conjunto de blocos lógicos configurados que possuem comportamento semelhante ao descrito.

Existem muitas implementações da arquitetura MIPS em HDL, mais especificamente VHDL (VHSIC (Very High Speed Integrated Circuit Hardware Description Language), porém essas implementações resumem-se apenas na implementação da arquitetura em si com um software embarcado diretamente — sem a possibilidade de incorporação de um código produzido externamente.

1.1 Problema

Segundo Albertini (2015), com a redução de custos de produção de FPGAs antigos com tecnologia de 65nm, muitos entusiastas, amadores e startups possuem agora a oportunidade de conseguir adquirir um dispositivo desse.

Com isso foram implementados vários trabalhos voltados para disciplinas de arquitetura de computadores, buscando uma forma de incentivar alunos a desenvolver hardware, porém as implementações existentes são limitadas interativamente e visualmente o que pode dificultar o aprendizado.

O desenvolvimento de uma ferramenta que permita utilizar recursos familiares aos alunos, como monitores e teclado, pode facilitar e fornecer a eles melhor acesso a esse tipo de implementação. Então, neste trabalho, é implementada uma ferramenta que poderia ser utilizada em aulas e também superar os limites de outras ferramentas.

1.2 Justificativa

Quando o primeiro dispositivo FPGA foi desenvolvido, havia muitas limitações devido à tecnologia da época, que não possibilitava a inserção de muitos blocos em um *chip*. Em consequência disso, era limitada à capacidade de sintetizar circuitos muito complexos, como uma arquitetura de um microprocessador. Com o avanço da tecnologia e a redução do tamanho dos componentes integrados, permitiu-se que os dispositivos FPGAs possuíssem milhares ou até milhões de blocos lógicos para sintetização de circuitos.

> FPGAs antigos, com poucas células lógicas e utilizando tecnologia de 65 nm, possuem projeto já consolidado e custo de produção baixo. Hoje é possível encontrar no mercado FPGAs com 10 K, elementos lógicos por USD 10, permitindo que startups, amadores e entusiastas tenham acesso a estes dispositivos[...]. (ALBERTINI, 2015, p.184).

Com a capacidade de sintetização ampliada, foram produzidas diversas implementações da arquitetura MIPS. Isso para tentar, de alguma forma, motivar alunos a produzirem *hardware* através desta tecnologia que, apesar de remota, somente agora está recebendo atenções das universidades.

As implementações existentes, embora consigam sintetizar completamente a arquitetura MIPS, são limitadas visualmente e interativamente, pois utilizam os displays integrados aos dispositivos de desenvolvimento e *switches* e botões para controlar a arquitetura. Devido a esta limitação, este trabalho se propõe a desenvolver uma interface programável e visual da arquitetura MIPS monociclo, utilizando o kit de desenvolvimento Altera-DE2 e/ou RZ EasyFPGA 2.2a, em conjunto com os dispositivos teclado e monitor, para melhor entendimento/assimilação da execução das instruções.

1.3 Objetivos

O principal objetivo deste trabalho é projetar uma ferramenta que permita: executar algumas instruções do MIPS (add, sub, and, or, slt, lw, sw, beq, j e addi) no modo monociclo, em um dispositivo de prototipação FPGA, programar e visualizar o fluxo de dados da arquitetura e os dados presentes na memória RAM, ROM e banco de registradores.

Ademais, para que o objetivo principal seja alcançado com sucesso, há determinados objetivos específicos que devem ser cumpridos. A realização dessas etapas é imprescindível para o avanço do projeto:

1. Desenvolver a arquitetura MIPS e um montador *Assembler*, em VHDL, que atendam as intruções: add, sub, and, or, slt, lw, sw, beq, j e addi;
2. Desenvolver um software para auxiliar no desenho da interface gráfica do ambiente;
3. Desenvolver controladores para envio de sinal de video com conexão VGA e para receber dados de um teclado com conexão PS/2;
4. Desenvolver um controlador e gerenciador para mémoria SDRAM externa ou desenvolver meios de utilizar blocos de memória internos para sintetizar RAM.
5. Desenvolver um editor de texto básico (somente armazenamento e exibição de texto) que consiga armazenar os dados recebidos do teclado para programação em Assembly;
6. Prover uma interface simples e intuitiva que permita visualizar:
   a) os dados utilizados pela arquitetura MIPS;
   b) o caminho de dados em que demonstre os caminhos utilizados pelas instruções;
   c) a área de programação em Assembly;
   d) a área de controle de execução e montagem do programa codificado.
7. Avaliar a execução das instruções, bem como a atualização dos dados no monitor.

1.4 Estrutura do Trabalho

O trabalho em questão está organizado da seguinte forma:

- As revisões bibliográficas serão apresentadas no capítulo 2, onde estão incluídas as revisões necessárias para conseguir compreender a execução e a complexidade deste trabalho e a apresentação de trabalhos relacionados a este e também;
- O capítulo 3 trata dos procedimentos metodológicos, onde estão contidos os métodos utilizados e os materiais necessários para reprodução da presente pesquisa;
- O desenvolvimento e a descrição das técnicas utilizadas serão fornecidos no capítulo 4, onde serão dadas informações necessárias para a implementação da ferramenta e a integração de outras arquiteturas ao projeto;
- O capítulo 5 explicita a avaliação da ferramenta e as dificuldades da implementação desse tipo de ambiente na tecnologia de FPGA;
- A conclusão é, por fim, tratada no capítulo 6, onde são apresentadas as inferências em relação aos resultados obtidos no desenvolvimento do projeto, além de subsidiar possíveis projetos que poderão ser implementados futuramente.

2 Revisão Bibliográfica

“O sábio que tudo sabe é aquele que sabe que nada sabe”.
Platão

Este capítulo tem como objetivo fornecer fundamentos para a total compreensão do trabalho, detalhando as principais tecnologias e conceitos utilizados, de forma a esclarecer de que forma os procedimentos citados no capítulo 3 foram realizados.

2.1 Fundamentos Teóricos

2.1.1 FPGA

O FPGA é um circuito digital integrado que visa integrar conceitos de circuitos CPLD’s (*Complex Programmable Logic Device*) e ASIC’s (*Application Specific Integrated Circuits*). O primeiro dispositivo que implementava este conceito foi criado pela Xilinx ® em 1984.

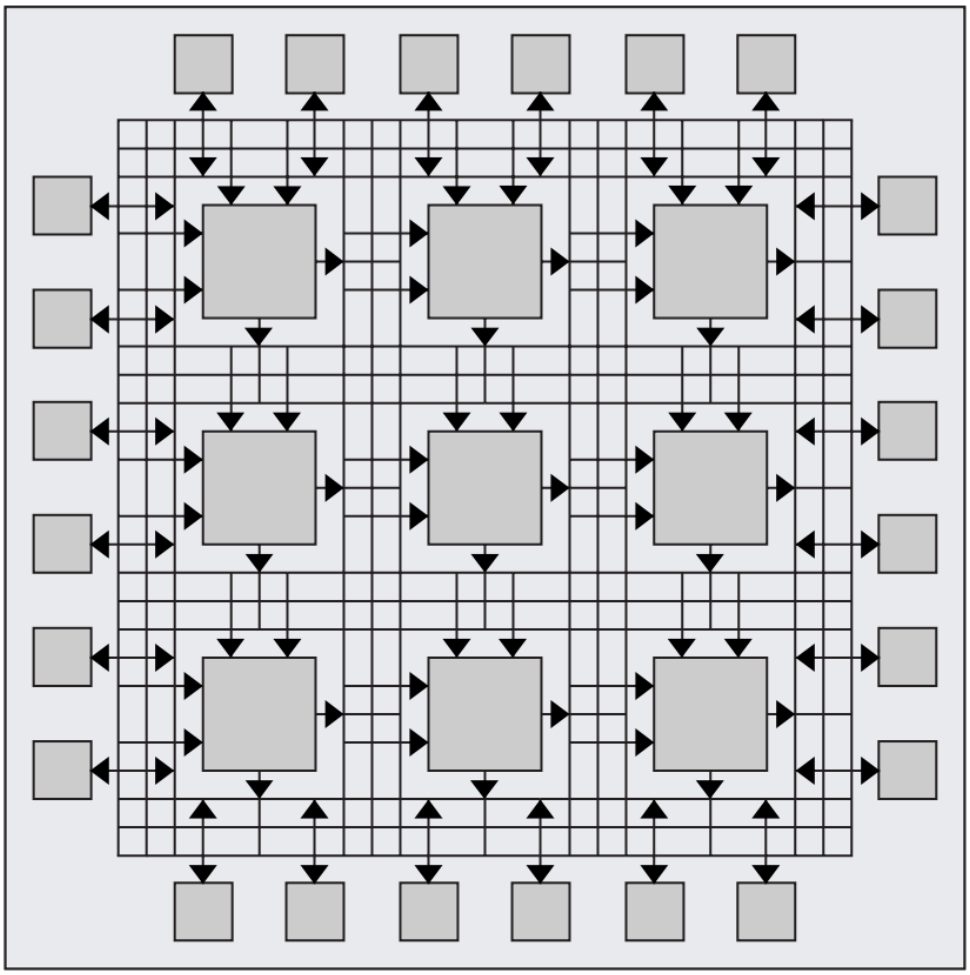
> Este dispositivo surgiu para preencher a lacuna que existia, ate então, no universo dos circuitos integrados digitais. De um lado do espectro encontravam-se os SPLD’s e CPLD’s, que eram extremamente configuráveis e possuíam tempos de projecto rápidos, mas não conseguiam implementar funções muito longas e complexas. Do outro lado do espectro existiam os Application-Specific Integrated Circuit (ASIC). Estes conseguiam suportar funções de extrema complexidade e de superior extensão, mas o seu projecto era excessivamente longo e dispendioso. Além disso assim que se fabrica o ASIC, o projecto fica estático no silício, ou seja, impossível de ser alterado. (OLIVEIRA, 2012)

Os CLPD’s e SPLD’s (*Simple Programmable Logic Device*), conforme citado por Oliveira (2012), são extremamente configuráveis e o tempo de desenvolvimento é menor. Para isso, o circuito interno é dividido em alguns blocos lógicos que podem ser configurados para representar algum tipo de porta lógica ou um registrador. Esses blocos são interconectados por uma matriz de interconexão, de forma que a informação possa fluir e seja sintetizado o circuito. A matriz pode levar a informação tanto para outros blocos lógicos e para blocos de entrada e saída de forma que possa ter comunicação com periféricos como memórias, controladores, conversores e dentre outros componentes eletrônicos.

Na figura 3, temos um exemplo de como esses blocos podem ser organizados dentro do *chip*. E é possível identificar os blocos lógicos e os blocos de entrada e saída se comunicando através da matriz de interconexão.

Para que o hardware seja configurado da maneira desejada, é utilizado um software que utiliza uma linguagem de programação como VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) ou Verilog. Essas linguagens descrevem o hardware e o software que, por sua vez, interpreta esse código, configura o dispositivo interligando e definindo como os blocos irão se comportar.

Figura 3 – Uma visão abstrata de um FPGA; células lógicas são embarcadas em uma estrutura geral de roteamento.



(HAUCK; DEHON, 2010, p.34, tradução nossa)

> *Field-programmable gate arrays* modernos possuem centenas de milhares de *lookup tables* (LUTs), centenas de memórias embarcadas, e centenas de multiplicadores conectadas através de um tecido de interconexão programável. Obviamente, é muito difícil programar o FPGA com a granularidade destes elementos individuais. Porém, com ferramentas de síntese e layout modernos, é possível descrever um modelo simples escrevendo expressões lógicas, em um nível mais alto que o nível de portas e a ferramenta faz o resto. (HAUCK; DEHON, 2010, p.129, tradução nossa)[1]

Ainda segundo Hauck e DeHon (2010), a forma mais popular de descrever o circuito é modelando a nível RTL (Register Transfer Level), pois permite ao arquiteto descrever o circuito fornecendo a lógica de comunicação entre pares de registradores. Isso torna o processo mais rápido, porquanto o arquiteto não precisará se preocupar em desenvolver o nível lógico de cada componente como um registrador, multiplexador, apenas se preocupando com a lógica

[1] Modern field-programmable gate arrays (FPGAs) contain hundreds of thousands of lookup tables (LUTs), hundreds of embedded memories, and hundreds of multipliers connected through a programmable interconnect fabric. Obviously it is intractable to program the FPGA at the granularity of these individual elements. However, with modern synthesis and layout tools, it is possible to describe a design simply by writing logical expressions, a level higher than gates, and letting the tools do the rest.(HAUCK; DEHON, 2010, p.129)

de comunicação entre os registradores. A linguagem VHDL é uma linguagem que suporta o desenvolvimento a nível RTL.

Dentre os recursos que possui, pode haver a presença de PLL's (*Phased-Lock Loop*). Um PLL, segundo Hsieh e Hush (1996), é um dispositivo que, contando com um sinal de referência, pode alterar sua frequência, assim como sua fase. O PLL no FPGA, pode ser utilizado, por exemplo, para multiplicar o clock aumentando ou diminuindo a frequência de entrada dependendo do hardware que o arquiteto estiver modelando. Alguns chips podem possuir mais de um PLL, com vários propósitos, logo podem ter diferentes clocks envolvidos em sua implementação.

Com a compreensão do funcionamento desta tecnologia, é possível perceber a sua ampla aplicação, dada a sua alta reconfigurabilidade, o avanço da tecnologia que permite inserir milhares de LUTS dentro de um chip, sua viabilidade econômica e redução do tempo de desenvolvimento. Tem-se, assim, um forte atrativo, quando comparado ao seu primeiro protótipo, que dispunha de uma arquitetura bastante limitada, com poucas LUTs disponíveis. Segundo Albertini (2015, p.185), a qualidade, as opções disponíveis e o preço estão seguindo o crescimento do mercado, diminuindo a distância entre um projeto em um dispositivo programável e um dedicado.

2.1.2 Autômatos Finitos

Os autômatos finitos (ou máquinas de estados finitos) são modelos matemáticos para representar a forma que um determinado sistema se comporta diante de uma determinada entrada. É uma máquina abstrata, formada por uma quantidade determinada (ou finita) de estados, podendo assumir, por assim dizer, apenas um estado por vez.

> Um Sistema de Estados Finitos é um modelo matemático de sistema com entradas e saídas discretas. Pode assumir um número finito e pré-definido de estados. Cada estado resume somente as informações do passado necessárias para determinar as ações para a próxima entrada. (MENEZES, 2000, p.32).

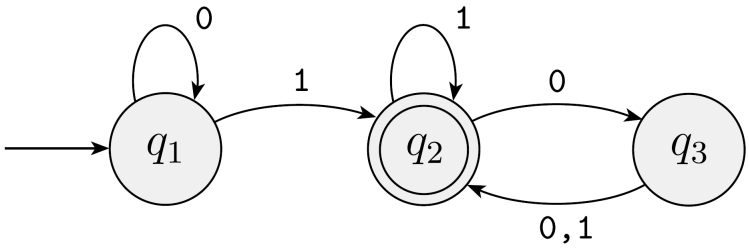
Os autômatos possuem algumas partes que são importantes para o compreendimento do que faz. Segundo Sipser (2013, p.35, tradução nossa), a definição formal diz que um autômato finito é uma lista de cinco objetos: conjunto de estados, entrada do alfabeto, regras de movimentações, estado inicial e estados de aceitação[2]. Quando se possui um sistema modelado matemáticamente, este pode ser definido em forma de diagrama de estados como na figura 4.

Na figura 4 pode-se observar todos cinco objetos citados por Sipser (2013):

- Estado: São os círculos simples identificados com um texto no meio, como q_1, q_2 e q_3;
- Estado de Aceitação: É o círculo duplo também identificado com texto no meio, como q_3.

[2] The formal definition says that a finite automaton is a list of those five objects: set of states, input alphabet, rules for moving, start state, and accept states. (SIPSER, 2013)

Figura 4 – Um autômato finito que possui três estados.

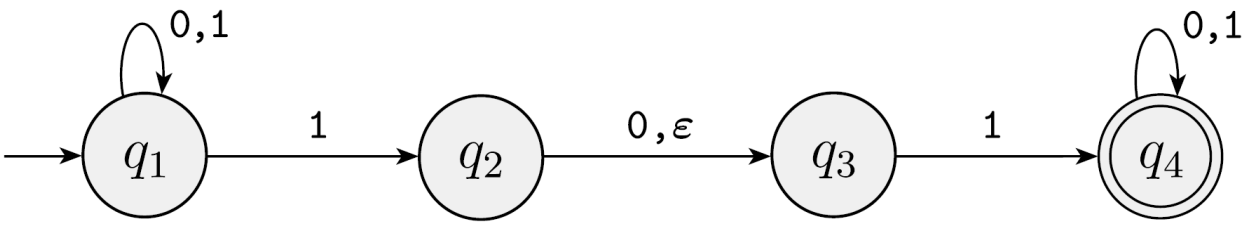


(SIPSER, 2013, p.34, tradução nossa)

- Estado Inicial: É o círculo apontado por uma seta também identificado com texto no meio, como q_1.
- Transição: As transições definem as regras de movimentações e são identificadas por setas indicando o estado anterior, o estado futuro e a condição para trocar de estado, como, por exemplo, na figura a seta que liga o estado q_1 ao estado q_2, indicando que para o autômato trocar do estado q_1 para o estado q_2 é preciso que receba a letra do alfabeto '1';
- Alfabeto: o alfabeto é identificado por todas as letras aceitas pelo autômato, no caso da imagem o alfabeto deste autômato é '0' e '1';

Os autômatos finitos podem ser classificados entre determinísticos (AFD) e não-determinísticos (AFN). Uma AFD é quando se sabe não apenas o estado atual da máquina, mas exatamente qual o estado futuro, de acordo com uma nova entrada da palavra lida. Por exemplo, ainda na figura 4 é perceptível que, quando a máquina está no estado q_1, se ela receber a letra '0", permanece no próprio estado ou, caso seja '1', troca de estado. Então, pode-se dizer que há um estado futuro determinado. Quando uma ou mais transições ocorrerem com uma mesma entrada, esta pode ser classificada como uma AFN, logo toda AFD é uma AFN.

Figura 5 – Um autômato finito não-determinístico.

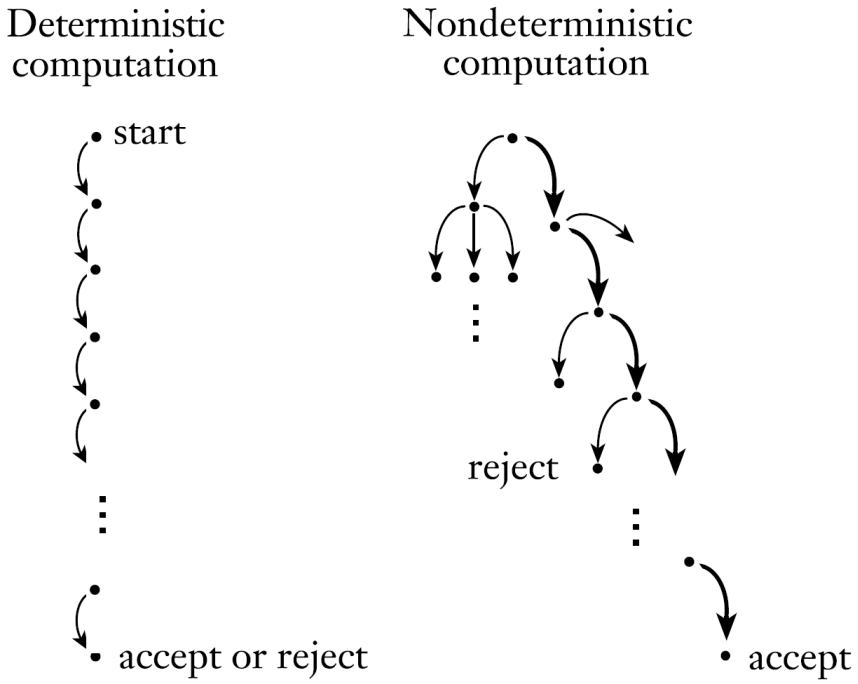


(SIPSER, 2013, p.48, tradução nossa)

Na figura 5, pode-se observar com clareza a ocorrência do não-determinismo. No estado q_1 há duas transições em que é aceita a letra '1' sendo possível transitar para o próprio estado q_1 ou para o estado q_2. Para verificar se uma palavra é aceita ou não por um AFN, são

feitos vários testes. No caso de uma AFD, é verificado apenas na última letra inserida, como pode ser observar na figura 6.

Figura 6 – Computação determinística e não-determinística com ramos de aceitação.



(SIPSER, 2013, p.49)

Tendo em vista que, para a tradução de um código de linguagem de programação para linguagem de máquina, é preciso que se leia cada letra, o uso de um autômato torna o processo facilitado e visualizável. Isso se explica devido à sequência de passos entre estados, os quais permitem definir aceitação e erro, para auxiliar o programador em seu desenvolvimento.

2.1.3 VGA

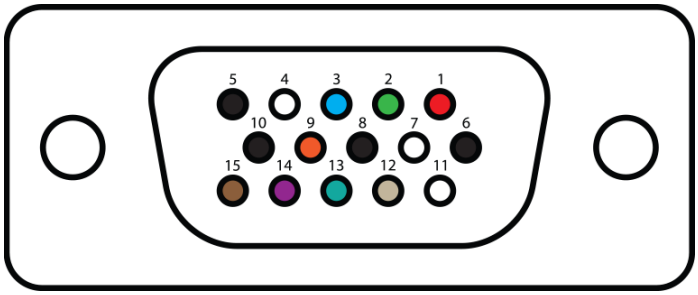
O VGA é um padrão desenvolvido pela IBM (Internation Business Machines), em 1987, para transmissão de imagens em tempo real. A comunicação é feita através do conector VGA com três linhas de cinco pinos. Na figura 7, encontra-se a identificação e localização de cada pino de comunicação e, na tabela 1[3] está a descrição de cada um.

> Um monitor VGA recebe a informação de cor com a ajuda de 3 sinais: R (vermelho), G (verde) e B (azul). Cada um destes sinais controla a emissão de electrões que por sua vez iluminam um ponto na superfície do ecrã com uma cor primária. Os níveis analógicos destes sinais especificam a intensidade de cada uma das cores primárias podendo variar entre 0 V (completamente escuro) e 0.7 V (brilho máximo). (SKLIAROVA, 2005, p.20)

Como dito por Skliarova (2005), um monitor recebe a informação de uma cor com base no padrão RGB que utiliza as cores vermelho, verde e azul como cores primárias. Com base na

[3] Os pinos que não foram citadas o seu uso nesta subseção (4, 9, 11, 12, 15), devido à inaplicabilidade da presente pesquisa, podem ser encontrados no site da VESA (*Video Electronics Standards Association*), que é a associação responsável por definir os padrões de exibição de video.

Figura 7 – Conector VGA com identificação dos pinos e cores representativas.



(WIKIPEDIA, 2019, Adaptado)

Tabela 1 – Lista de pinos com numeração referente à figura 7

Pino	Descrição	Pino	Descrição
1	Red Video (Analógico, 0-0.7V)	9	+5V
2	Green Video (Analógico, 0-0.7V)	10	Sync Ground
3	Blue Video (Analógico, 0-0.7V)	11	Monitor ID Bit 0
4	Reserved	12	DDC Serial Data Line
5	Ground	13	Horizontal Sync
6	Ground Red	14	Vertical Sync
7	Ground Green	15	DDC Data Clock Line
8	Ground Blue		

(PINOUTS, 2019, Adaptado)

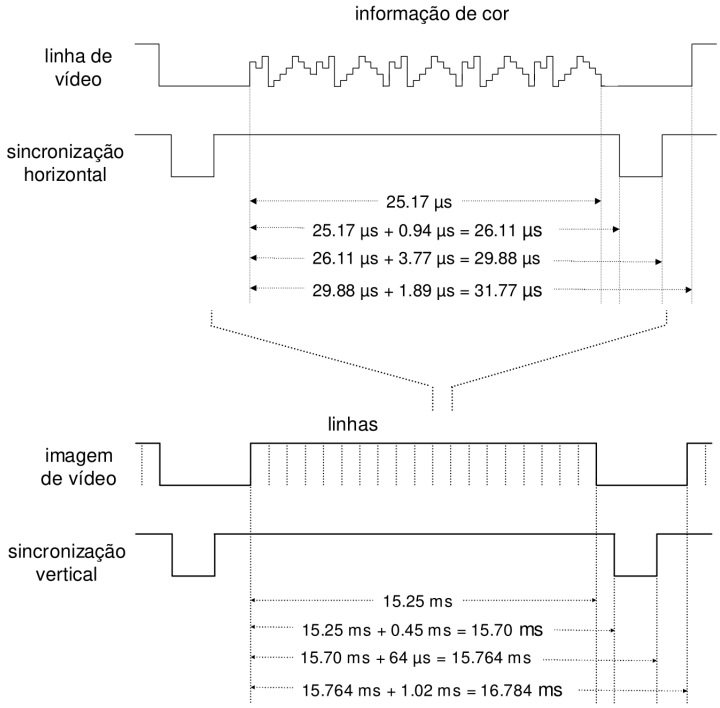
variação da tensão de entrada dessas cores, obtém-se outras. Numericamente, essas cores em computadores são representadas por 8 bits, 256 intensidades diferente, pada cara uma. Isto permite produzir um total de 16.777.216 cores.

A imagem no monitor tanto de LCD (*Liquid Crystal Display*) quanto de CRT (*Catode-Ray Tube*), a imagem é sempre desenhada de cima para baixo e da esquerda para a direita e, para isso, há a necessidade de se detectar quando é preciso saltar a linha ou voltar para a parte superior, para o qual há os sinais de sincronia horizontal (pino 13) e vertical (pino 14). O sinal horizontal indica quando começam e terminam as linhas da imagem. De de maneira análoga, o sinal vertical indica quando as colunas começam e terminam, iniciando uma nova imagem.

Na figura 8, é possível observar a forma de onda durante a sincronização. A resolução de tela utilizada na fonte da figura é de 640 colunas por 480 linhas. Analisando a figura, pode-se observar a presença de espaço de tempo entre a troca de linha ou de coluna. O tempo anterior à exibição da imagem é chamado *back porch* (BP) e possibilita ao monitor apontar a posição de desenho novamente para o início da linha. Após o término da área visível, é chamado *front porch* (FP) e tem o propósito de separar o momento em que é feita a troca de linha e a ativação do sinal de sincronia (HSync e VSync). A esse momento de sincronia, é dado o nome de *sync pulse* (SP), que é o tempo (ou ciclos de clock) em que o sinal de sicronia ficará ativo.

Todas as contagens de tempo são dadas em ciclos de clock, logo, para completar uma

Figura 8 – Características temporais dos sinais de sincronização.



(SKLIAROVA, 2005)

linha na área visível, é preciso que aconteçam 640 pulsos de clock, dentro de determinado tempo, para que o monitor entenda que há 640 colunas. Os padrões de temporização de linhas e colunas são feitos pela VESA (*Video Eletronics Standards Association*) e na tabela 2 encontra-se alguns desses valores[4].

Tabela 2 – Temporização de diferentes resoluções.

Resolução		Intervalos de Sincronização						Taxa de Atual. (TA)	Pixel Clock
Colunas	Linhas	HBP	HFP	HSP	VBP	VFP	VSP		
640	480	40	8	96	33	10	2	60 Hz	25.175 MHz
640	480	120	16	64	16	1	3	75 Hz	31.5 MHz
800	600	88	40	128	23	1	4	60 Hz	40 MHz
1024	768	160	24	136	29	3	6	60 Hz	65 MHz
1280	720	220	110	40	20	5	5	60 Hz	31.5 MHz
1360	768	256	64	112	18	3	6	60 Hz	31.5 MHz
1920	1080	148	88	44	36	4	5	60 Hz	148.5 MHz

(VESA, 2007, Adaptado)

O *pixel clock* (PC) é a frequência necessária para o envio dos pixels ao dispositivo, incluindo quantas imagens (com a resolução escolhida) serão exibidas por segundo (TA), como pode-se observar na tabela 2 em que a taxa de atualização mais frequente é a de 60 Hz. Essa frequência pode ser calculada pela seguinte equação:

[4] HBP: *Horizontal Back Porch*; HFP: *Horizontal Front Porch*; HSP: *Horizontal Sync Pulse*; VBP: *Vertical Back Porch*; VFP: *Vertical Front Porch*; VSP: *Vertical Sync Pulse*; TA: Taxa de atualização de imagens na tela

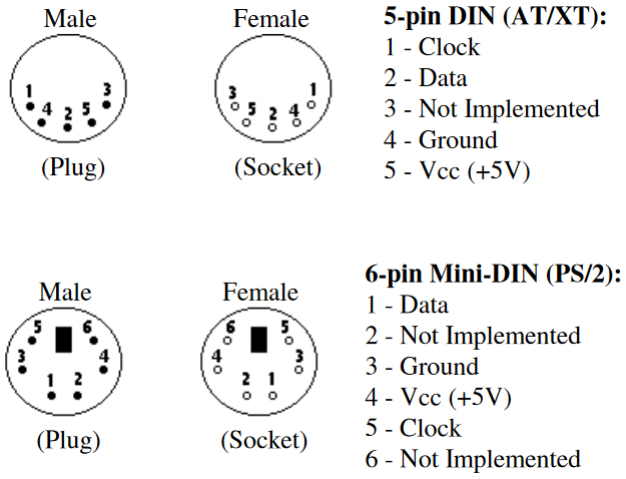
$$(Colunas + HBP + HFP + HSP) * (Linhas + VBP + VFP + VSP) * TA = PC$$

A cada pulso de clock é enviada uma cor para o monitor que, com base nos sinais de sincronização, exibe a cor na posição, de acordo com a contagem de tempo interno. Ao mesmo tempo, ele consegue descobrir a resolução do monitor, de acordo com os sinais de sincronia. Com isso, a imagem é exibida e atualizada de acordo com as entradas.

2.1.4 PS/2

O PS/2 é um conector que pode ter dois estilos: O DIN, com 5 pinos, e o mini-DIN, com 6 pinos. Ele foi desenvolvido pela IBM ®, para permitir conectar alguns teclados e mouses compatíveis. Os dois estilos podem se comunicar entre si, contanto que seja utilizado um adaptador.

Figura 9 – Pinagem de cada conector. (Adaptado)



(CHAPWESKE, 2003, p.2)

Segundo Chapweske (2003, p.3), a comunicação via PS/2 é bidirecional, síncrona e serial. O sinal está ocioso quando o *clock* e o sinal de dados estão ativos em alto. O periférico é que produz o sinal de *clock*, logo é esse dispositivo o responsável pela sincronização.

Os dados são sempre transmitidos em um *byte* por vez e todos são enviados dentro de uma palavra de 11 bits. Essa palavra pode ser visto na tabela 3.

Tabela 3 – Corpo da mensagem durante a comunicação. (Adaptado)

Inicio	Dados	Paridade	Fim da mesnsagem	Reconhecimento
1 bit	8 bits	1 bit	1 bit	1 bit
Observação: O bit menos significativo de dados é o primeiro bit recebido.				

(CHAPWESKE, 2003, p.3)

> O teclado envia pacotes de dados, scan codes, para o *host* indicando que a tecla foi pressionada. Quando a tecla é pressionada ou mantida pressionada um *make code* é transmitido. Quando é liberada um *break code* é enviado. Toda tecla está relacionada a um código único de parada e da tecla, então o host pode determinar exatamente o que aconteceu. (RUDRAMMA; KRIHNA, 2013, p.155, tradução nossa)[5]

O *make code* é um código de identificação da tecla que está pressionada e o *break code* é o código para identificar quando a tecla foi liberada. A quantidade de bytes enviados pelo teclado depende da tecla pressionada Algumas demandam um código maior e outras menor. Na tabela 4, é possível observar alguns dos códigos enviados pelo teclado, de acordo com suas teclas. Os códigos estão escritos em hexadecimal, para melhor observância dos fatos.

Tabela 4 – Tabela de tradução dos *scan codes* de teclados. (Adaptado)

Tecla Base	Shift + Tecla Base	Código da Tecla	Código de Parada
`	~	0E	F00E
1	!	16	F016
2	@	1E	F01E
3	#	26	F026
4	$	25	F025
5	%	2E	F02E
6	^	36	F036
7	&	3D	F03D
8	*	3E	F03E
9	(	46	F046
0	)	45	F045
Alt Direito		E011	E0F011
Alt Esquerdo		11	F011

(VETRA, 2019)

Com base na tabela 4, pode-se observar que o código de parada sempre terá o código 'F0', antes do código da tecla. O código 'E0' é utilizado somente em caso de teclas estendidas, que possuem o mesmo código de tecla de outra tecla, como é o caso dos 'Alt's, os quais possuem o mesmo código, mas, para diferenciá-los, é inserido o 'E0', antes de enviar o código do 'Alt' direito.

Esses códigos são importantes para poder compreender como o teclado se comunica através de uma porta PS/2, facilitando a tradução desse código para algum outro padrão, seja a execução de um comando específico ou mesmo a conversão para outro código, como, por exemplo, o ASCII (*American Standard Code for Information Interchange*) que é um código de troca de informações americano por meio de sinais de 8 bits.

[5] The keyboard sends packets of data, scan codes, to the host indicating which key has been pressed. When a key is pressed or held down a make code is transmitted. When a key is released a break code is transmitted. Every key is assigned a unique make and break code so that the host can determine exactly what has happened.(RUDRAMMA; KRIHNA, 2013, p.155)

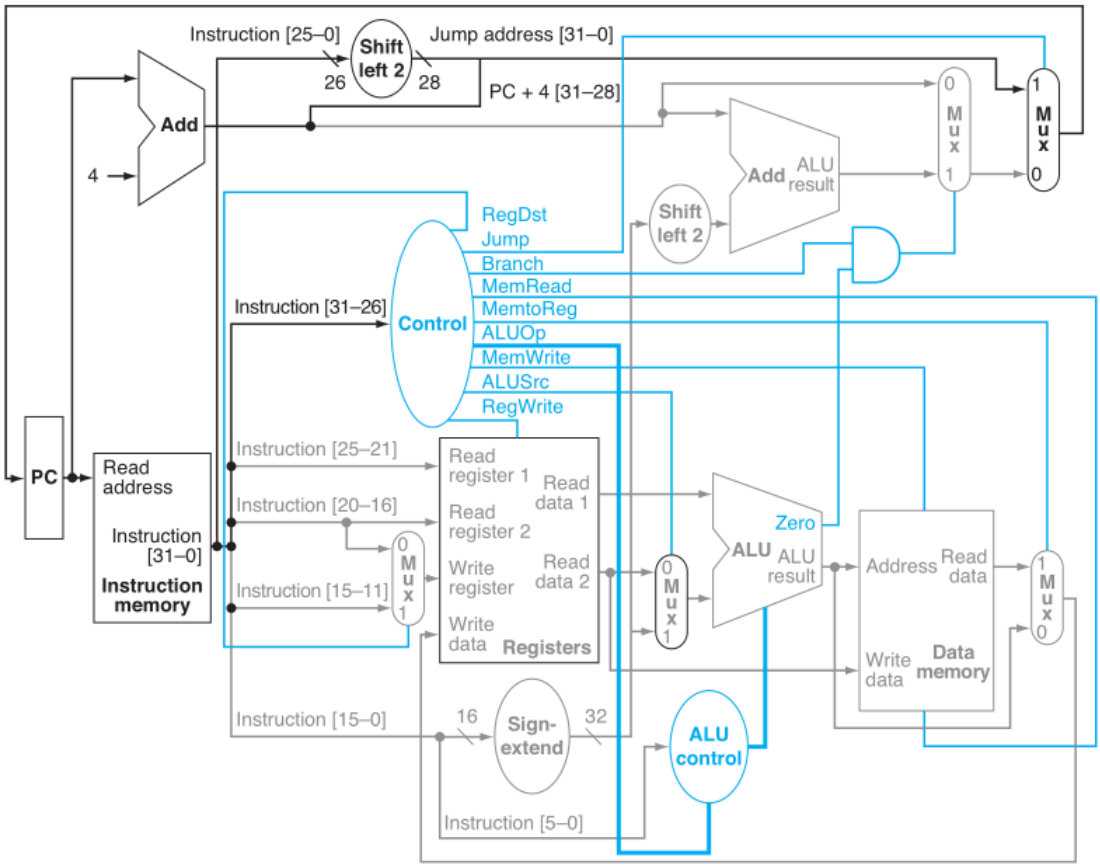
2.1.5 Arquitetura Monociclo MIPS

Uma arquitetura de computador tem por função demonstrar a forma com que os componentes estão organizados e como devem se comunicar entre si e, assim, executar as instruções para a qual foi planejado. Existem muitas arquiteturas desenvolvidas ao longos dos anos, como a MIPS (1981) ou Intel® 8086 (entre 1976 e 1978).

Cada arquitetura tem sua forma de trabalhar e todas dependem de ciclos de *clock* para marcar o passo-a-passo de execução das instruções. Algumas precisam de mais de um ciclo e outras apenas um, os quais podem ser nomeados em multiciclo e monociclo, respectivamente. Exemplificando, a arquitetura MIPS se enquadra neste tipo de execução, devido à simplicidade e maleabilidade de sua arquitetura. Há também algumas melhorias possíveis, como a inserção de pipelines para a execução sobreposta de instruções, porém esta faz com que o modelo fique cada vez mais complexo, dependendo da quantidade de pipelines.

A arquitetura MIPS é uma arquitetura desenvolvida por Hennessy e Patterson (2008), e o livro escrito por eles, "*Computer Organization and Design*", é uma das referências de muitos professores para o ensino da disciplina de Arquitetura de Computadores. Dentre as suas características, segundo Penha, Fontes e Ferreira (2016), estão: poucas instruções, decodificação direta, registradores gerais, pipeline, load/store como unidade de execução.

Figura 10 – Caminho de dados e controle simples para tratamento de instruções de salto.



(PATTERSON; HENNESSY, 2012, p.329)

A arquitetura Monociclo MIPS, objeto desta pesquisa, foi desenvolvida com embasamento no livro de Hennessy e Patterson (2008), onde são fornecidos detalhes importantes para o desenvolvimento da arquitetura em si, contudo de forma simplória e funcional, com poucas instruções e de maneira compreensível.

A figura 10 é o modelo de caminho de dados do monociclo MIPS, apresentado no livro. Analisando a figura, observa-se que há extruturas necessárias para a sua implementação. Utiliza-se memória RAM para uso geral, memóra ROM (*Read-Only Memory*) para instruções, somadores, banco de registradores, ALU (*Arithmetic Logic Unit*), multiplexadores entre outros componentes que podem ser encontrados na figura. É possível perceber também a execução de uma instrução, não sendo utilizadas todas as conexões para a sua realização. No caso, as linhas pretas são os sinais de fluxo, os azuis sinais de controle e cinza é a parte inativa.

Das estruturas utilizadas, a única que demanda mais tempo do arquiteto durante a implementação é a estrutura de controle de sinais, pois é nela que são interpretados os comandos recebidos das instruções e são enviados para gerenciar os outros componentes, que são, em sua maioria *chips*, integrados prontos para uso, como, por exemplo, a memória RAM e o banco de registradores.

As instruções a serem executadas são palavras binárias, as quais são dividas em fragmentos de onde são extraídos os sinais que servem para identificá-las. Elas podem possuir tamanhos variados, o MIPS, por exemplo, utiliza 32 bits por palavra e estas ficam armazenadas na memória ROM, também chamada como memória de instruções.

> A memória de instruções precisa providenciar somente o acesso de leitura, pois o caminho de dados não escreve instruções. Como a memória de instruções é somente de leitura, nós a tratamos como combinação lógica: a saída a qualquer momento reflete o conteúdo do local especificado pelo enderenço de entrada e nenhum sinal de controle é necessário.(HENNESSY; PATTERSON, 2008, p.293, tradução nossa)

As instruções possuem formatos diferentes, na maneira em que são fragmentadas, de forma a padronizar e simplificar o processo de controle. Esses fragmentos são chamados campos e possuem funções específicas, como, por exemplo, identificar os registradores a serem utilizados e qual a função a ser executada. Na arquitetura MIPS, por exemplo, o formato R pode-se dividir da seguinte forma, segundo Hennessy e Patterson (2008, p.63):

Figura 11 – Divisão de campos de instruções do tipo R nomeados

(HENNESSY; PATTERSON, 2008, p.63)

- *op*: Operação básica da instruções, também conhecida por *opcode*;
- *rs*: Primeiro registrador de origem da operação;

- *rt*: Segundo registrador de origem da operação;
- *rd*: Registrador de destino da operação, ele recebe o resultado da operação.
- *shamt*: Quantidade de shift, não é usado então recebe o valor '0';
- *funct*: Função. Campo que seleciona uma variação específica da operação do campo *op* e às vezes chamado de *function code* (código da função);

> Os designers do MIPS fizeram o compromisso de manter todas as instruções com o mesmo tamanho, necessitando, assim, de diferentes formatos de instruções para diferentes tipos de instruções. Por exemplo, as instruções acima são chamadas tipo R (para registrador) ou tipo-R. Um segundo tipo de formato de instruções é chamado tipo I (para imediato) ou tipo-I e é usado com números imediatos e instruções de transferência de dados.(HENNESSY; PATTERSON, 2008, p.293, tradução nossa)

Conforme explicado por Hennessy e Patterson (2008), desenvolvedores planejaram o MIPS para possuir instruções de apenas um tamanho, 32 bits, apenas fazendo divisão dos campos dentro da instrução definindo formatos. Estes formatos podem ser encontrados na tabela 5 e pode ser observado que o tamanho é mantido, mas o seu propósito é diferente conforme os comentários na última coluna.

Tabela 5 – Formatos de instruções do MIPS.

Nome	Campos						Comentários
Tamanho do Campo	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções do MIPS de 32 bits
Tipo-R	op	rs	rt	rd	shamt	funct	Formato de instruções aritméticas
Tipo-I	op	rs	rt	endereço/imediato			Transferência, desvio, formato de imediato
Tipo-J	op	endereço destino					Instrução de salto

(HENNESSY; PATTERSON, 2008, p.104, tradução nossa)

As instruções são escritas em binário e de difícil entendimento e, para facilitar a programação para o MIPS, foi criada a linguagem *Assembly* MIPS, que é uma linguagem com nível mais próximo do hardware, mas não é uma linguagem a nível lógico. Pode-se observar que o *opcode* possui 6 bits, o que permite a arquitetura possuir até 64 instruções e se combinados com o campo *funct* poderia possuir, caso exista apenas instruções do tipo R, até 4096 instruções.

Após programado, um montador monta o código a nível de máquina identificando os tipos de instruções, registradores, imediatos, endereços e operações. A seguir segue um exemplo de como são escritas as instruções nos formatos descritos na tabela 5:

- Tipo R: *op rd, rs, rt*
- Tipo I:

  - Desvio: *op rd, rs, endr*
  - Imediato: *op rd, rs, imm*
  - Transferência: *op rd, imm(rs)*

- Tipo J: *op endr*

Os registradores utilizados pelas instruções estão presentes em um banco de registradores, o MIPS utiliza um banco com 32 registradores de 32 bits, sendo que na posição *0* não é possível escrever, apenas ler o valor *0*.

O PC (Program Counter) é um registrador utilizado para armazenar a posição para selecionar as instruções na memória. Quando é feito um salto, o PC é alterando diretamente pelo programa permitindo que possa executar outra área da memória.

> Com a ideia de programa armazenado, é necessário ter um registrador para segurar o endereço da instrução que está executando. Por razões históricas, este registrador é quase sempre chamado contador de programa, abreviado por PC, na arquitetura MIPS, embora o nome adequado poderia ser registrador de endereço de memória.(HENNESSY; PATTERSON, 2008, p.80, tradução nossa)

Quando a instrução é selecionada e enviada pela memória de instruções, ela é desmembrada e processada de acordo com o seu *opcode* e *funct*. O processo ocorre prevendo todos os formatos previamente citados. No caso, os de tipo R, I e J. O *opcode* são enviado para o bloco de controle, local em que serão definidos os sinais de ativação dos outros blocos. O *funct* é enviado ao controle da ULA (Unidade lógico-aritmética) ou ALU (*Arithmetic Logic Unit*), para definir qual a função que irá exercer como subtração, comparação ou adição.

Ao mesmo tempo é separado os 16 bits menos significativos para o processamento de instruções do tipo I. Esse sinal é extendido e pode ser utilizado tanto como endereço para saltos quanto valores imediatos para realizar alguma operação na ULA. O sinal como endereço é somado ao PC.

Durante a separação dos campos, também é separado o campo de endereço de 26 bits de instruções do tipo J. Com esse campo, é identificada a área da memória em que o PC irá apontar no próximo ciclo de clock. Por ser uma instrução específica para salto, é importante ser um sinal de barramento muito grande como 26 bits, pois isso garante o acesso a grande parte da memória.

A cada pulso de clock é completado um ciclo na arquitetura, onde é somado '1' ou atribuído algum valor ao endereço da memória de instruções, para execução de novas instruções. Este ciclo se repete até que acabem as instruções, ou que as próprias instruções produzam um *loop* infinito, ou a própria contagem das posições de memória resetem após a contagem, ultrapassar o tamanho máximo da memória.

2.1.6 *Synchronous Dynamic Random Access Memory* (SDRAM)

A memória RAM (*Random Access Memory*) é um tipo de memória em que os dados permanecem armazenados enquanto a memória estiver energizada, ao contrário da memória

ROM (*Read-only Memory*), em que os dados permanecem armazenados mesmo sem energia. Dependendo da forma em que ela é fabricada, como, por exemplo, utilizando capacitores, os dados podem ser perdidos, mesmo que energizada, pois, como no exemplo, eles se descarregam depois de um determinado tempo.

> A memória contém uma máquina de controle interna que deve ser inicializada corretamente, antes de entrar em modo de operação normal. Depois que a memória é energizada e que os níveis corretos de tensão sejam atingidos, uma sequência de comandos deve ser enviada à memória para garantir seu correto funcionamento. (BONATO, 2008, p.22)

Existem alguns tipos de memórias como DRAM (RAM Dinâmica), SRAM (RAM estática) e SDRAM (RAM Síncrona Dinâmica), as quais possuem diferenças em seu funcionamento. As DRAMs e SDRAMs são de fácil fabricação e grande densidade, porém são lentas e seus dados são armazenados apenas por um tempo e se perdem após um tempo, mesmo energizadas. As SRAMs são memórias que utilizam um circuito para manter os dados armazenados. Estas últimas, por possuírem um circuito mais complexo, têm menor densidade, mas o desempenho é muito alto e seus dados não se perdem até que sejam desenergizadas.

> Memórias SRAM existem desde os anos 60, e memórias DRAM desde os anos 70. Ao contrário do que o nome sugere, a DRAM não é caracterizada pela rapidez e, sim, pelo baixo custo, aliado à alta capacidade, em comparação com a SRAM. Isso se deve ao fato das suas células de memória serem mais simples, o que torna possível criar chips com maior número de células de memória. Em compensação, o mecanismo de acesso às suas células de memória é mais complicado. (VASCONCELOS, 2002, p.649)

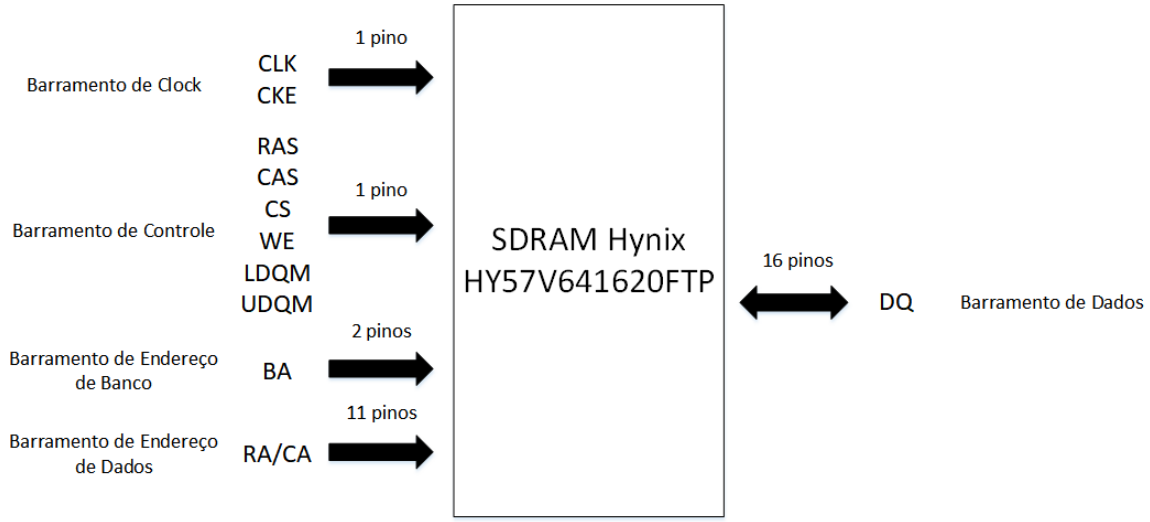
A SDRAM é uma DRAM com o diferencial de permitir ao processador que se sincronizem através do uso de um sinal de *clock*. Os componentes internos entre si não são diferentes,mas a forma em que os dados são acessados, tornando a SDRAM mais veloz que a DRAM.

> Essas memórias usam um velho truque para permitir acessos em um único ciclo. Este truque é utilizado pelas placas de vídeo gráfico, desde os anos 80. Dentro de um chip de memória SDRAM, existem 4 bancos de memória independentes. Quando são acessadas, as células de mesmos endereços em cada um dos 4 bancos internos do chip são acessadas. Terminado o primeiro acesso (digamos que este primeiro acesso demore 5 ciclos, portanto a memória estaria operando com a temporização 5-1-1-1), o dado do primeiro banco poderá ser transmitido ao chipset e ao processador, e os três dados dos outros três bancos poderão ser transmitidos imediatamente depois, sem ter que esperar pelo seu tempo de acesso tradicional. A demora está em chegar aos dados desejados. Uma vez acessados, podem ser rapidamente transmitidos. (VASCONCELOS, 2002, p. 441)

Na figura 12 pode-se observar a presença de algumas portas as quais estão agrupadas, simbolicamente, em barramentos para melhor compreensão de suas funções. Baseado no *datasheet* disponibilizado pela Hynix ®, tem-se as seguintes descrições:

- Barramento de Clock: Possui a função de receber o sinal de *clock* (CLK) e habilitar a sua entrada no *chip*, através do sinal de *Clock Enable* (CKE).

Figura 12 – Bloco de memória SDRAM, da Hynix® HY57V641620FTP, apresentando e agrupando os pinos presentes no chip.



(HYNIX, 2007, p.3, Adaptado)

- Barramento de Controle: Possui a função de controlar o chip para executar os comandos internos, com um pino para cada porta. De acordo com Hynix (2007):
  - $\overline{RAS}$: Possui a função de informar ao *chip* quando está sendo enviado o endereço da linha em que o dado está localizado.
  - $\overline{CAS}$: Possui a função de informar quando está sendo enviado o endereço da coluna em que o dado está localizado.
  - $\overline{CS}$: Habilita ou desabilita todas as entradas, com exceção de: CLK, CKE, UDQM e LDQM.
  - $\overline{WE}$: Habilita ou desabilita a escrita do dado na memória.
- Barramento de Endereço dos Bancos: Tem a função de seleciona o banco em que o dado está localizado. A Hynix® HY57V641620FTP possui 4 bancos, tornando necessário 2 pinos para o seu endereçamento.
- Barramento de Endereço de Dados: É utilizada para informa qual a linha (RA, *Row Address*) e coluna (CA, *Column Address*) em que o dados está localizado dentro do banco.
- Barramento de Dados: É composto apenas pela porta DQ que utiliza 16 pinos ($DQ0$ $DQ15$). Esse barramento é de entrada e saída, pelo qual os dados trafegarão durante a sua leitura ou gravação.

2.1.6.1 Inicialização

Como a SDRAM possui uma máquina de controle interno, é preciso que essa máquina esteja com a eletricidade estabilizada. Ela também deve estar configurada antes de realizar os

comandos de leitura e escrita. Para ser configurada, é necessário executar uma sequência de comandos que garanta o seu funcionamento.

Segundo Bonato (2008, p.23), após a energização da memória e que o relógio do circuito esteja estável, é necessário aguardar o intervalo de 200ms, antes de executar qualquer comando na memória. E, para isso, segundo Li e Wu (2013, p.2233, tradução nossa), a inicialização normal consiste de uma espera de 200μs, e executar os comandos *precharge-all-banks*, *auto-refresh*, e *mode-register-set*.

O comando *mode-register-set* é o principal comando da inicialização, através do qual são passadas informações de como a memória deve funcionar. Com ela é possível configurar o seguinte:

- *Write Mode*: Define se a quantidade de palavras a serem escritas é apenas uma (*Single Write*) ou a quantidade definida pelo *Burst Length* (*Burst Write*);
- *CAS Latency*: Define quantos ciclos de *clock* serão necessários aguardar, após ativar o banco, para executar outro comando ou receber os dados da memória.
- *Burst Type*: Informa qual a forma que é feita a leitura e a escrita, sem sequência ou intercalado.
- *Burst Length*: Define quantas *words* serão passadas durante a leitura ou escrita, caso Write Mode esteja configurado em *Burst Write*, da memória.

Tabela 6 – Formato do comando *Mode Register Set* da memória Hynix® HY57V641620FTP. (Adaptado)

BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	Write Mode	0	0	CAS Latency			Burst Type	Burst Length		

(HYNIX, 2007, p.6)

Na tabela 6, pode-se observar as posições em que a memória identifica, nos barramentos de endereços, a configuração a ser realizada internamente.

Com as informações passadas, a memória irá se comportar na devida forma e nos devidos tempos, os quais devem ser respeitados pelo controlador para que a comunicação com a memória seja correta.

2.1.6.2 Leitura e Gravação

Para que a memória consiga executar a leitura e a gravação de dados, é preciso que seja ativada a linha em que será feita a leitura dos dados, por meio do comando *Bank Active*. Para que isso seja possível, é passada a linha através do barramento de endereço e o banco em que será ativada essa linha.

Após a solicitação de execução deste comando, a memória somente receberá um novo comando após a latência do CAS (*CAS Latency*) findar-se, assim ela permitirá receber um

Tabela 7 – Combinação de sinais necessários para executar o comando *Bank Active* na memória. (Adaptado)

Command	CAS	RAS	CS	WE	A[11:0]	BA[1:0]
Bank Active	1	0	0	1	RA	BA

(HYNIX, 2007, p.12)

novo comando. No comando, conforme mostrado na tabela 7, observa-se que é enviada a linha que será ativa, devido a todas as entradas de controle possuírem sinal invertido. Com isso, RAS e CS estão ativadas, ao contrário de CAS e WE. Os sinais RA e BA são, respectivamente, os sinais do endereço da linha e do endereço do banco.

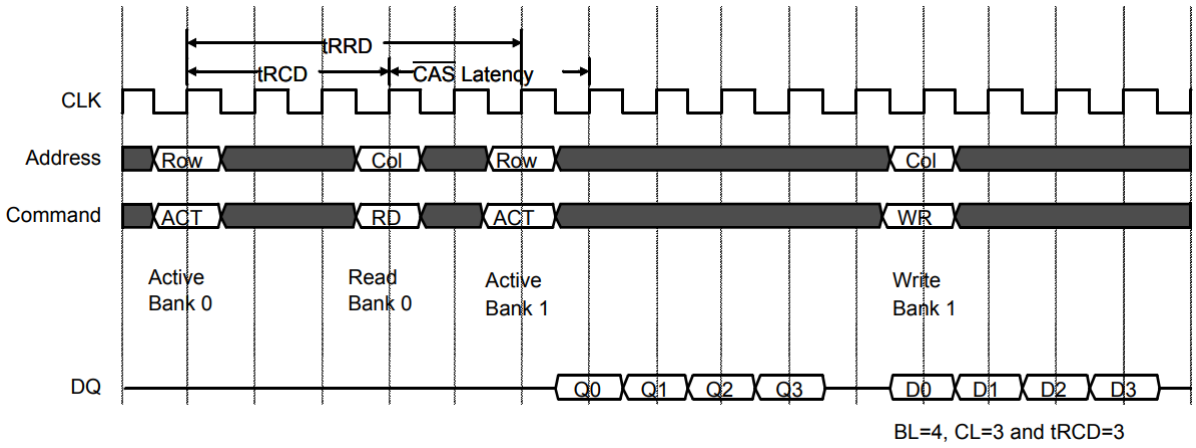
Tabela 8 – Combinação de sinais necessários para executar o comando *Bank Active* na memória. (Adaptado)

Command	CAS	RAS	CS	WE	A[7:0]	A10	BA[1:0]
Read	0	1	0	1	CA	0	BA
Write	0	1	0	0	CA	0	BA

(HYNIX, 2007, p.12)

A partir do momento em que o banco e a linha estão ativados e o tempo de latência finalizado, basta executar os comandos de leitura, informando a coluna em que se encontra a informação. A memória ficará ocupada novamente, sendo necessário aguardar mais uma vez a latência, mas dessa vez somente para a leitura. A gravação começa durante o envio dos sinais de escrita. Observa-se tal fato na figura 13. Os sinais para informar o comando à memória podem ser encontrados na tabela 8, onde CA é o endereço da coluna.

Figura 13 – Formas de onda demonstrando a temporização de execução dos comandos *Bank Active Read* e *Write*.



(HYNIX, 2003, p.2)

Na figura 13, observa-se a temporização em que os comandos devem ser executados, para que seja possível a gravação e leitura dos dados da memória. É exibido, primeiramente, a ativação da linha, em seguida é feito o comando de leitura, aguardando o tempo de latência do CAS que é de 3 *clocks*. Também é possível visualizar que o *Burst Length* (BL) está em 4, realizando a leitura de 4 palavras, os quais não possuem latência entre si. As mesmas coisas ocorrem para a escrita, mas pode ser observado que o banco, o qual receberá os dados, foi ativado antes de fazer a leitura. As informações de envio de comando se encontram no *datasheet* do *chip*, informando o tempo entre comandos.

2.1.7 Montador (*Assembler*)

As linguagens de programação são a forma utilizada para a comunicação com o computador.Estes se comunicam através de sinais lógicos (0 e 1), o que é chamado de linguagem de máquina. Programar, utilizando linguagens de máquina, é trabalhoso e demorado, por isso foi criada a linguagem *Assembly*, que é uma representação simbólica do nível de máquina.

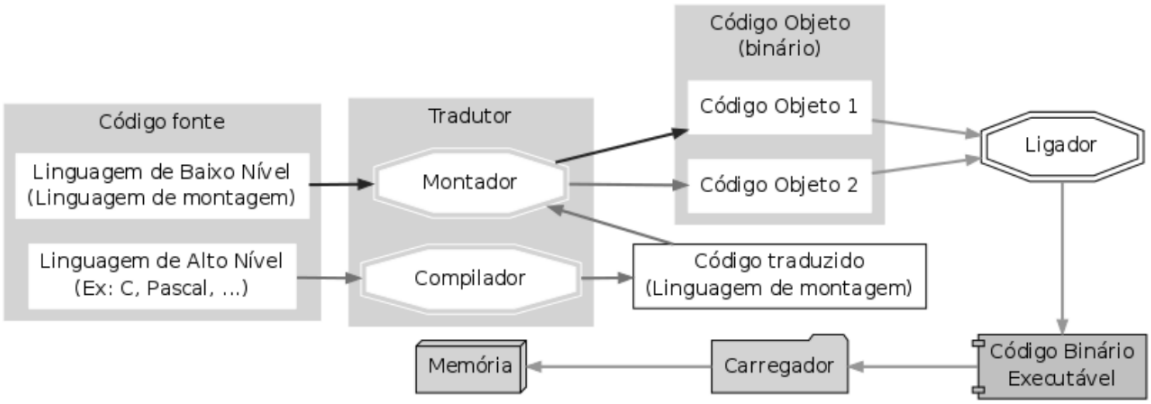
O *Assembly* é a linguagem mais próxima do hardware, e pode é classificada como linguagem de montagem e de baixo-nível. Existem linguagens que se aproximam da linguagem humana, o que possibilita a melhor compreensão cód igo scrito classificadas ficadas como linguagem de alto-nível, como Java, C, Python, entre outras.

Todas as linguagens precisam ser traduzidas para os sinais lógicos, para que o computador consiga compreender as instruções. Para isso, existem os tradutores. As linguagens de baixo nível são traduzidas diretamente para o nível lógico, devido à sua proximidade na forma com que a máquina se comunica. Por sua vez, as linguagens de alto nível são traduzidas para o baixo nível.

> Os tradutores podem ser divididos em dois grupos, dependendo da relação existente entre a linguagem fonte e a linguagem alvo. Quando a linguagem fonte for essencialmente uma representação simbólica para uma linguagem de máquina numérica, o tradutor é chamado de montador e a linguagem fonte é chamada de linguagem de montagem. Quando a linguagem fonte for uma linguagem de alto nível, como é o caso do Pascal ou do C, e a linguagem alvo for uma linguagem de máquina numérica ou uma representação simbólica desta linguagem (linguagem de montagem), o tradutor é chamado de compilador. (FARIAS; MEDEIROS, 2013, p.65)

Na figura 14, observa-se as diferenças de tradução entre um compilador e um montador. Nota-se que o compilador é dependente de um montador para traduzir seu código para a linguagem de máquina. Após traduzido, são gerados códigos objetos, já em linguagem de máquina, que são ligados para formar um código executável. A máquina consegue, agora, ler as instruções traduzidas e carregá-las na memória para sua execução.

Figura 14 – Passos no processo de compilação



(SOUSA; JúNIOR; FORMIGA, 2014, p.14)

2.2 Fundamentos Históricos

"Não há fatos eternos, como não há verdades absolutas.".
Friedrich Nietzsche

A arquitetura MIPS é abordado em vários artigos em que são implementados simuladores em softwares e a própria arquitetura em FPGA. Esta seção apresenta dois trabalhos em que são desenvolvidos simuladores desta arquitetura, em software e descritos em hardware.

2.2.1 Um Ambiente Gráfico de Simulação Baseado no MIPS em VHDL

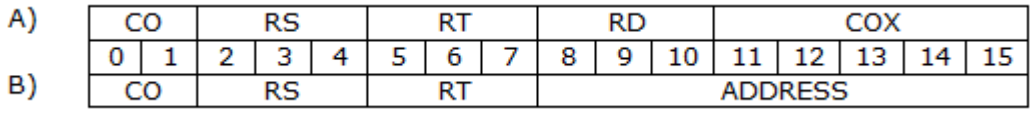
No trabalho "*A Visual Simulation Environment for MIPS Based on VHDL*", os autores Llorente, Pulido e Rubio (2000) desenvolveram um ambiente de simulação gráfico em VHDL que facilita a interpretação dos dados numéricos obtidos através das simulações.

> Existem muitas ferramentas modeladas para produzir código VHDL, com um resultado de simulação, em uma sequência de estados muito abstratos.Para os estudantes que estão começando a conhecer como uma CPU com *pipeline* funciona, dificilmente conseguem interpretar.(LLORENTE; PULIDO; RUBIO, 2000, p.56, tradução nossa)

Devido à versão do compilador utilizada, software VSIM *simulator*, possuir algumas limitações, a arquitetura MIPS teve que ser reduzida, o que proporciona um conjunto de instruções e capacidade de memória menor. O banco possui 8 registradores de 16 bits (2 *words*) e a memória de dados 256 palavras de 8 bits, equivalendo a 128 palavras de 16 bits. Isso permite que o endereçamento seja feito diretamente pelo campo de endereço. São suportadas dois tipos de instruções: Tipo-R (*register-to-register*) e instruções de memória.

A figura 15 mostra como é a divisão de campos das instruções de 16 bits. O campo CO é o código da operação que identifica o tipo da instrução, quando o valor é "00"quer dizer que é instruções *register-to-register*, que possui o campo COX, de 5 bits, para indicar a operação que irá ocorrer. O endereçamento de registradores é de 3 bits. As instruções suportadas são

Figura 15 – Formatos de instruções: A) *register-to-register*, B) memória.



(LLORENTE; PULIDO; RUBIO, 2000, p.58)

ADD, SUB, AND, OR e SLT. Já as instruções de memória possuem um campo para o endereço (ADDRESS) de 8 bits, suportando as instruções LW, SW e BEQ.

Para programar para a arquitetura, eles providenciaram um montador para uma linguagem de montagem que criaram. Segundo Llorente, Pulido e Rubio (2000, p.59), as linhas poderiam possuir as seguintes possibilidades:

- Linha em branco, que é ignorada;
- Comentários, que começa com o sinal --"e terminam somente com quebra de linha;
- Definir variáveis (DB ou DW, um identificador e, opcionalmente, um valor);
- Definir uma *label* (nome da *label*, seguida de ":");
- e, uma instrução de montagem.

O analisador gráfico desenvolvido, na figura 16, permite o estudo visual do comportamento da máquina frente às instruções executadas. Ele permite observar as linhas que interligam os blocos da arquitetura e, assim, acompanhar o fluxo de dados.

A figura 16 exibe a tela do analisador gráfico, em que permite visualizar os dados através das linhas. Ele permite controlar a velocidade de execução, avançar ou retornar o estado e visualizar qual instrução está sendo executada no momento. Ele permite visualizar os dados das linhas variando a sua tonalidade e exibindo etiquetas com o valor.

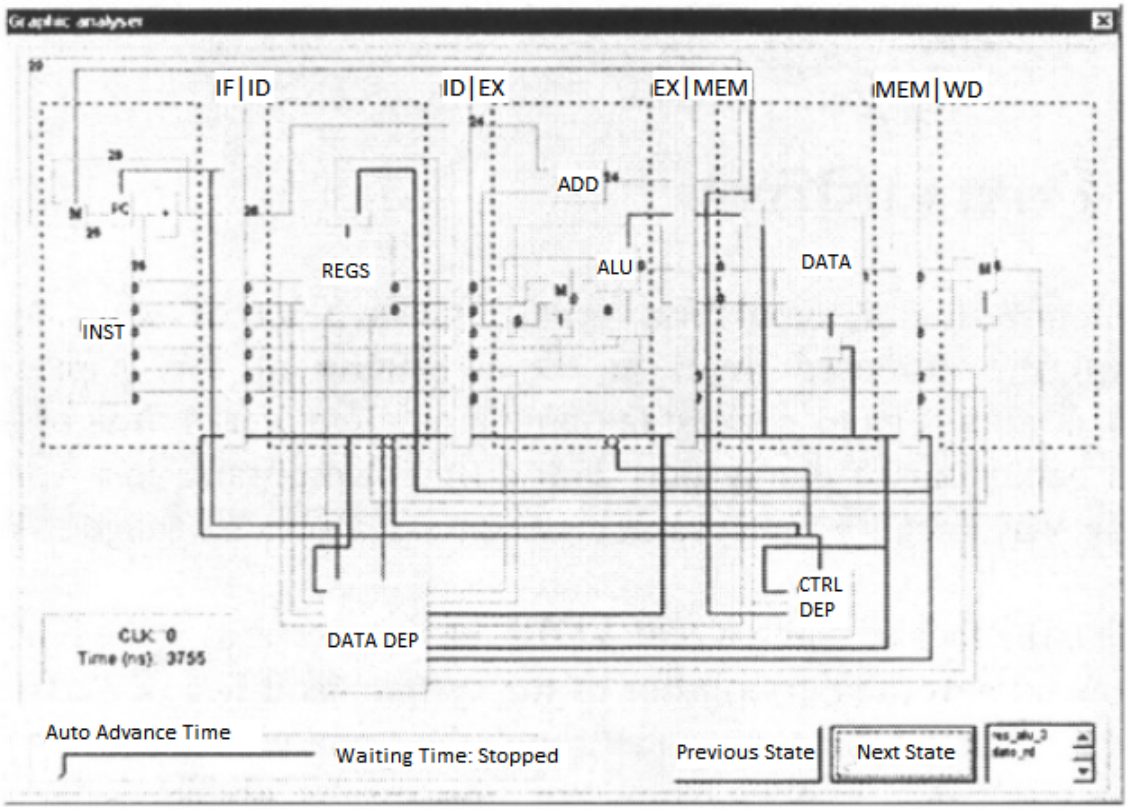
A simulação inicia com a construção da simulação do MIPS em VHDL. Após construído, é necessário inserir o código do programa a ser simulado e este salva o resultado em arquivos. Através do VSIM da *Model Technology Inc.*, os dados são coletados e extraídos os resultados, os quais são exibidos no caminho de dados da arquitetura, a cada estado da máquina.

A conclusão dos autores foi que um ambiente visual permite entender como o pipeline trabalha. É possível explorar os comportamentos do hardware, utilizando programas desenvolvidos rapidamente, através da linguagem de montagem que a arquitetura utiliza.

2.2.2 MIPSFPGA - Um Simulador MIPS Incremental com Validação em FPGA

No trabalho, os autores Penha, Fontes e Ferreira (2016) desenvolvem um simulador incremental, que permite a validação e prototipação do circuito desenvolvido em uma placa de desenvolvimento FPGA. Este simulador permite ao aluno:

Figura 16 – Janela de simulação.



(LLORENTE; PULIDO; RUBIO, 2000, p.61)

- Alterar o projeto da arquitetura para testes ou resoluções de exercícios de algum livro.
- Exportar o projeto da arquitetura para a sintetização do circuito em um dispositivo FPGA.
- Depurar o conteúdo dos registradores e das memórias durante simulação e prototipação.
- Desenvolver novos componentes para o simulador.

O simulador é desenvolvido com base no Hades, que é um simulador de circuitos em diversos níveis, permitindo que o projetista tenha liberdade no seu desenvolvimento.

O simulador incremental possui um conjunto de projetos e componentes para serem executados no simulador Hades. Esses projetos seguem a sequência de desenvolvimento do livro de Hennessy e Patterson (2008), em que são incrementadas partes da arquitetura para que possibilite atender a determinadas instruções.

Para que o simulador possa ser bastante interativo, os autores desenvolveram um componente de perguntas de múltipla escolha com base, também, no livro de Hennessy e Patterson (2008). Eles adaptaram as perguntas para que elas sejam de múltipla escolha. Alguns destes exercícios, presentes nos livros, solicitam ao aluno incrementar à arquitetura uma nova instrução e para isso criaram uma forma de estender o ISA (Instruction Set Archtecture) através de componentes. O aluno poderá incluir um novo componente, o framework Hades, que

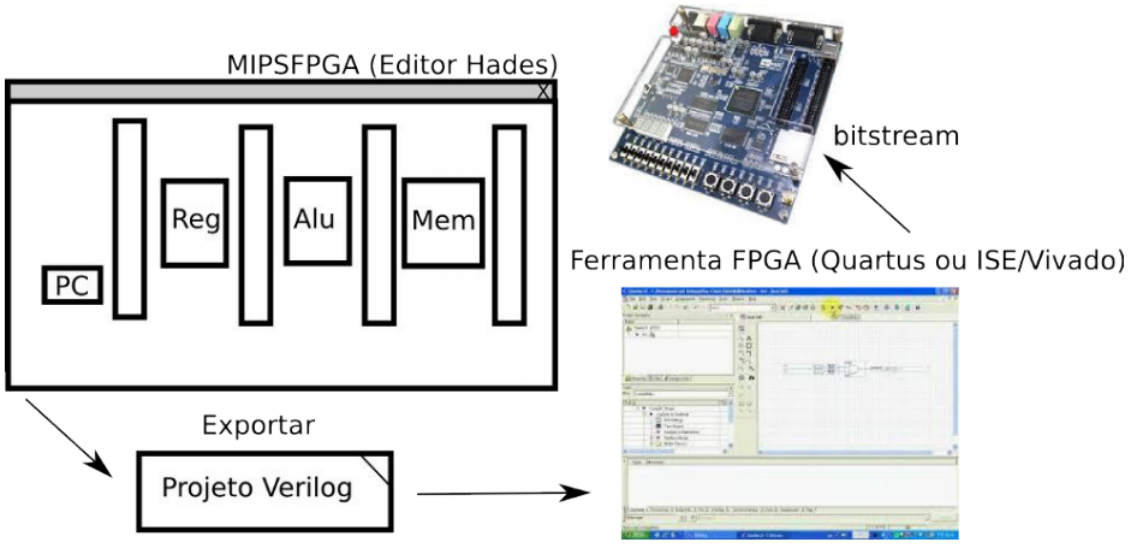
permite desenhar em tempo de execução, projetando-o e incluindo-o no circuito do projeto. Com isso, poderá testar se o circuito está funcionando de forma adequada.

O MIPSFPGA não é proposto para programar em *assembly*, porém foi inserida a funcionalidade que permite visualizar as instruções executadas durante os estágios do *pipeline* através da decodificação das instruções em binário.

> Os programas de simulação são ferramentas importantes para auxílio no aprendizado, porém a implementação física do projeto, com a prototipação em FPGA, permite ao estudante ter uma comprovação do funcionamento. (PENHA; FONTES; FERREIRA, 2016, p.23)

Os autores tiveram o propósito de permitir ao aluno prototipar e testar a arquitetura em algum *kit* de desenvolvimento FPGA, exportando para alguma linguagem de descrição de hardware, pois, segundo eles, são linguagens complexas, as quais as comunidades de programadores não estão habituados a programar por não serem imperativas como nas linguagens de programação como C, Java e C++. O simulador foi programado para exportar para a linguagem Verilog - com a mesma metologia utilizada, pode-se exportar para VHDL - e, ao término da exportação, o projeto está pronto para ser carregado em uma ferramenta que irá fazer a síntese em um FPGA. O processo pode ser observado na figura 17.

Figura 17 – Fluxo para exportação da implementação para prototipação em FPGA.



(LLORENTE; PULIDO; RUBIO, 2000, p.23)

Durante os testes no dispositivo FPGA, os autores proveram meios de validar a implementação, durante a exportação do projeto para Verilog. Incrementaram meios de possibilitar controlar o circuito externamente, através de botões e *switches* e também a visualização da instrução por meio de *displays* de sete segmentos.

Os autores concluem, mostrando os diferenciais do artefato que desenvolveram em relação às ferramentas existentes. Destacaram a capacidade de exportar para FPGA, ser simples, genérico e expansível, além do apoio ao ensino, ao incluir questões relacionadas à arquitetura.

3 Materiais e Métodos

“Um bom começo é a metade”.
Aristóteles

Para a produção do presente trabalho, foram realizadas pesquisas em livros, disponíveis na biblioteca da própria instituição, relacionados ao desenvolvimento de arquiteturas de computadores e o uso da linguagem VHDL. Embasou-se, também,em artigos técnicos, teóricos, científicos e dissertações disponibilizados pelo portal Periódicos Capes por intermédio do buscador interno e pelo Google Acadêmico

3.1 Materiais

Para o desenvolvimento da ferramenta proposta neste trabalho, foram utilizados os seguintes recursos:

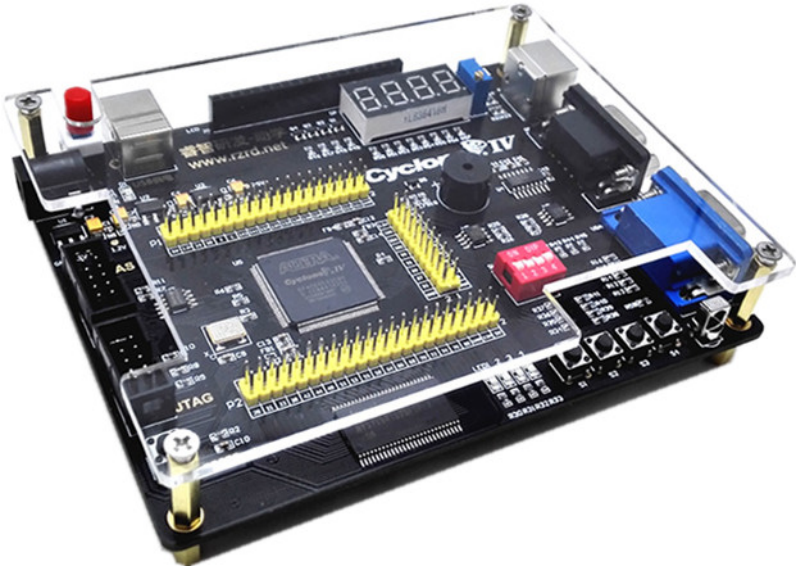
- Notebook com as seguintes especificações:
  - Modelo: Samsung Expert X30 2016 (NP500R5HE)
  - Processador: Intel Core i5-5200U CPU @2.20GHz 2.70GHz 3MB L3 Cache x86-64
  - Memória RAM: 8GB LPDDR3 @1600MHz
  - Placa de Vídeo: Nvidia Geforce 940M Graphics com 2GB DDR3
  - Sistema Operacional: Microsoft Windows 10 Pro de 64 bits

Figura 18 – Notebook utilizado durante o desenvolvimento da ferramenta.

(SAMSUNG, 2015)

- Placas de desenvolvimento FPGA:
  - EasyFPGA v2.2a
    - Modelo: EasyFPGA v2.2a
    - Chip: Intel Altera Cyclone IV E
      - Modelo: EP4CE6E22C8N;
      - Blocos Lógicos: 6272 LEs;
      - Quantidade de Entrada e Saída para usuário: 91;
      - PLL (*Phased-locked Loop*) de uso geral: 2;
      - Tensões Suportadas (Volts): 1.2, 1.5, 1.8, 2.5, 3.3;
      - Clock Máximo: @ 472.5 MHz;
      - Blocos de memória: 270Kb;
    - Memória RAM:
      - Modelo: Hynix HY57V641620FTP-7;
      - Capacidade: 64Mbit (67,108,864 bit);
      - Bancos: 4 x 1M;
      - Largura do barramento de dados: 16 bits;
      - Clock Máximo: @ 143 MHz;
      - 4096 ciclos de *refreshes* / 64 ms
    - Interfaces de periféricos e recursos:
      - 1x Botão de alimentação;
      - 1x Botão de reset;
      - 4x Botões de uso geral;
      - 4x switches de uso geral;
      - 4x LEDs (*Light Emitting Diode*);
      - 4x *Displays* de 7 Segmentos;
      - 1x Interface PS/2;
      - 1x Interface VGA;
      - 1x Receptor de infravermelho;
      - Dentre outros recursos;
  - Altera DE2
    - Modelo: Altera DE2-35C
    - Chip: Intel Altera Cyclone II:
      - Modelo: EP2C35F672C6;
      - Blocos Lógicos: 33,216 LEs;
      - Quantidade de Entrada e Saída para usuário: 475;
      - PLL (*Phased-locked Loop*) de uso geral: 4;
      - Tensões Suportadas (Volts): 1.2, 1.5, 1.8, 2.5, 3.3;

Figura 19 – Placa de desenvolvimento EasyFPGA v2.2a utilizada na produção desta pesquisa.



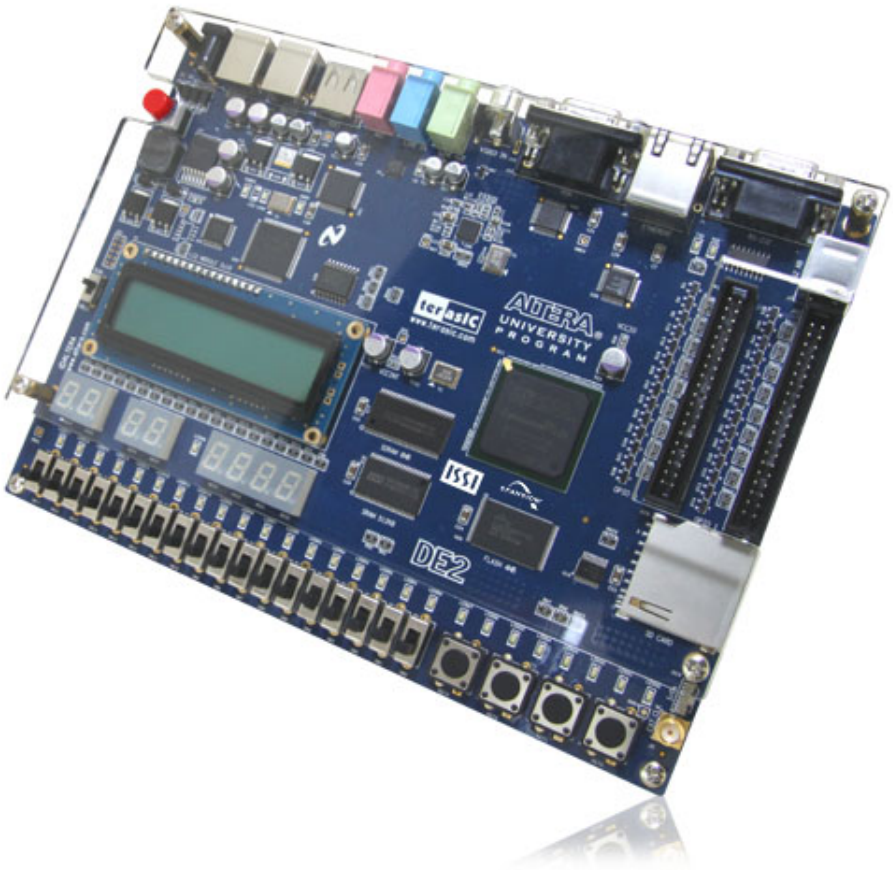
(RZRD, 2018)

- Clock Máximo: @ 260 MHz;
- Blocos de memória: 484 Kb;

* Memória RAM:
  - Modelo: Zentel A3V64S40ETP-G6;
  - Capacidade: 64Mbit (67,108,864 bit);
  - Bancos: 4 x 1M;
  - Largura do barramento de dados: 16 bits;
  - *Clock* Máximo: @ 166 MHz;
  - 4096 ciclos de *refreshes* / 64 ms
* Interfaces de periféricos e recursos:
  - 1x Botão de alimentação;
  - 1x Botão de *reset*;
  - 4x Botões de uso geral;
  - 18x *Switches* de uso geral;
  - 18x LEDs (*Light Emitting Diode*) de cor vermelha;
  - 9x LEDs (*Light Emitting Diode*) de cor verde;
  - 8x *Displays* de 7 Segmentos;
  - 1x Interface PS/2;
  - 1x Interface VGA;
  - Entre outros recursos;

A escolha de duas placas se justifica considerando a hipótese da mais simples, a EasyFPGA v2.2a, não comportar o projeto, devido a algum problema durante o desenvolvimento,

Figura 20 – Placa de desenvolvimento Altera DE2-35C utilizada na produção desta pesquisa.



(TERASIC, 2006)

quando então poderá ser substituída pela placa Altera DE2 para finalizar o desenvolvimento da ferramenta.

- Ambientes de desenvolvimento:
  - Altera Quartus Lite 13.0 Service Pack 1;
  - Altera ModelSim;
  - Netbeans 8.1;
  - Software Hades.
- Linguagens de Desenvolvimento:
  - Oracle Java;
  - VHDL 2008.
- Monitor com resolução de 1360x768;
- Teclado com comunicação PS/2;

3.2 Classificação

No que se refere aos seus objetivos, esta pesquisa pode ser classificada como pesquisa exploratória e experimental, uma vez que tem como finalidade implementar um ambiente de desenvolvimento inteiramente descrito em hardware e sua prototipação. Segundo Engel e Silveira (2009), "este tipo de pesquisa tem como objetivo proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a construir hipóteses".

Quanto à sua natureza, ela pode ser classificada como pesquisa básica, por utilizar de recursos e conhecimentos pré-existentes sem especificação de uso da ferramenta. Em outras palavras, como Engel e Silveira (2009, p.34) cita, a pesquisa ”objetiva gerar conhecimentos novos, úteis para o avanço da Ciência, sem aplicação prática prevista. Envolve verdades e interesses universais”.

Em relação à abordagem, classifica-se como pesquisa qualiquantitativa. Nesse sentido, a análise quantitativa é feita com base no uso de blocos do chip FPGA, de acordo com o desenvolvimento do ambiente (como a produção de controladores de vídeo e de memória RAM). Por sua vez, a análise qualitativa é identificada por meio da dificuldade com que a informação é encontrada para a produção dos controladores.

> A pesquisa quantitativa, que tem suas raízes no pensamento positivista lógico, tende a enfatizar o raciocínio dedutivo, as regras da lógica e os atributos mensuráveis da experiência humana. Por outro lado, a pesquisa qualitativa tende a salientar os aspectos dinâmicos, holísticos e individuais da experiência humana, para apreender a totalidade no contexto daqueles que estão vivenciando o fenômeno. (POLIT; BECK; HUNGLER, 2004, p.201).

No que tange aos procedimentos, pode ser classificado como estudo de caso, uma vez que se desenvolve uma ferramenta em um ambiente pouco explorado, procurando demonstrar singularidades que se tem ao se aplicar, neste mesmo ambiente, a tecnologia de FPGA.

> Um estudo de caso pode ser caracterizado como um estudo de uma entidade bem definida como um programa, uma instituição, um sistema educativo, uma pessoa, ou uma unidade social. Visa conhecer em profundidade o como e o porquê de uma determinada situação que se supõe ser única em muitos aspectos, procurando descobrir o que há nela de mais essencial e característico. (FONSECA, 2002, p.33).

3.3 Procedimentos Metodológicos

Os procedimentos realizados para a produção deste trabalho encontram-se na figura 21.

Figura 21 – Fluxograma das etapas deste trabalho.

(Fonte: O Autor)

4 Construção do Ambiente e Programação da Ferramenta para a Arquitetura MIPS Monociclo

“Uma experiência nunca é um fracasso,
pois sempre vem demonstrar algo”.
Thomas Edson

Ao se desenvolver um *hardware*, é preciso planejar toda a sua estrutura, como os componentes eletrônicos deverão se comunicar entre si e quais componentes serão utilizados. Neste capítulo, será descrita a maneira como foi planejado e desenvolvido o *hardware* do ambiente proposto no capítulo 1.

Para que o ambiente seja desenvolvido, é imprescindível projetar os componentes necessários para a sua comunicação, tanto externa como interna. Para a comunicação externa, é preciso criar controladores que consigam realizar o interfaceamento com os componentes que possuem um padrão de comunicação definido, como o da presente pesquisa. Assim, tem-se: a saída VGA, a entrada PS/2 e, talvez, a entrada e saída de dados da memória SDRAM, devido à sua complexidade de implementação, podendo ser substituído pelo uso de blocos de memória. Todos esses componentes são controladores para comunicar externamente do *chip* FPGA.

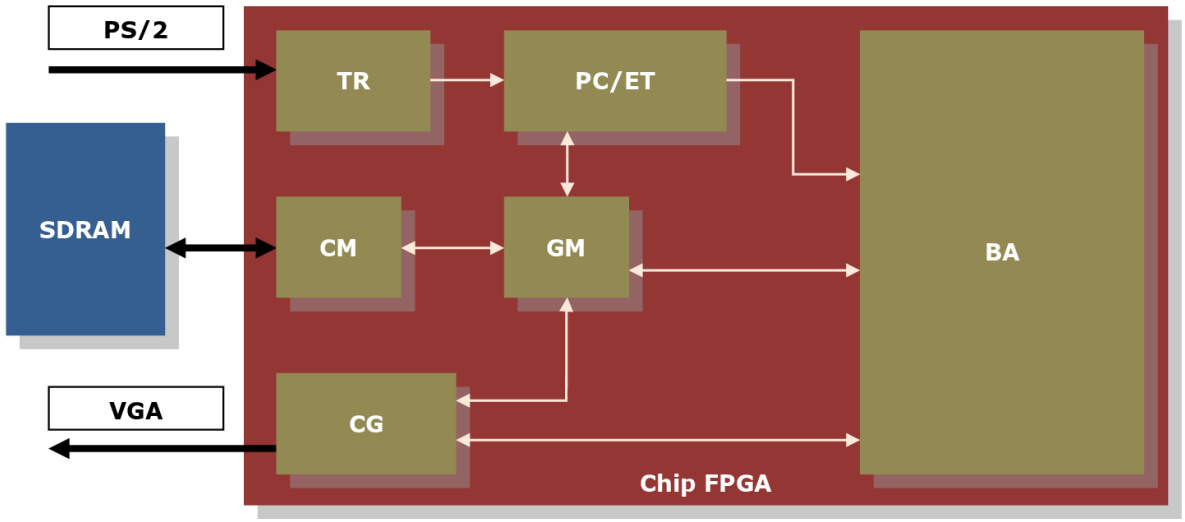
Esta implementação busca fazer a separação da arquitetura, de forma que seja apenas monitorada e buscadas informações para exibição diante das instruções que executar.

Internamente, são necessários componentes que venham a produzir e interpretar as informações recebidas pelo usuário, através de um teclado conectado à entrada PS/2. Isso permite controlar a arquitetura MIPS presente internamente, para a sua monitoração e coleta de dados, para exibição na tela.

Na figura 22, pode-se observar a arquitetura projetada para a presente pesquisa, a fim de demonstrar o que está proposto. Nela é possível observar os blocos internos, nos quais estão distribuídas as funções, atendendo aos requisitos propostos no capítulo 1. As descrições das siglas são as seguintes:

- TR (Tradutor): Bloco responsável por tratar a entrada de dados através da porta PS/2. Ela recebe os dados, traduzindo-os para o dado referente à tabela ASCII;
- ET (Editor de Texto): Bloco responsável por gerenciar o código-fonte enviado pelo usuário através do tradutor.
- PC (Processador de Comandos): Processador de comandos enviados pelo usuário;

Figura 22 – Arquitetura proposta para o desenvolvimento do ambiente em FPGA.



(Fonte: O Autor)

- CM (Controladora de Memória): Controladora responsável por se comunicar com a memória SDRAM para solicitações de escrita, leitura e *refresh*;
- GM (Gerenciador de Memória): Bloco responsável por realizar solicitações à controladora de memória de acordo com um uma fila de prioridade pré-definida;
- CG (Controladora Gráfica): Bloco responsável por enviar os sinais de vídeo da interface gráfica para a saída VGA;
- BA (Bloco de Arquitetura): Este bloco reúne as partes de gerenciamento, exibição e controle da arquitetura desenvolvida, diante do fato desta pesquisa estar envolvido o desenvolvimento do MIPS, o bloco contém o controle do mesmo.

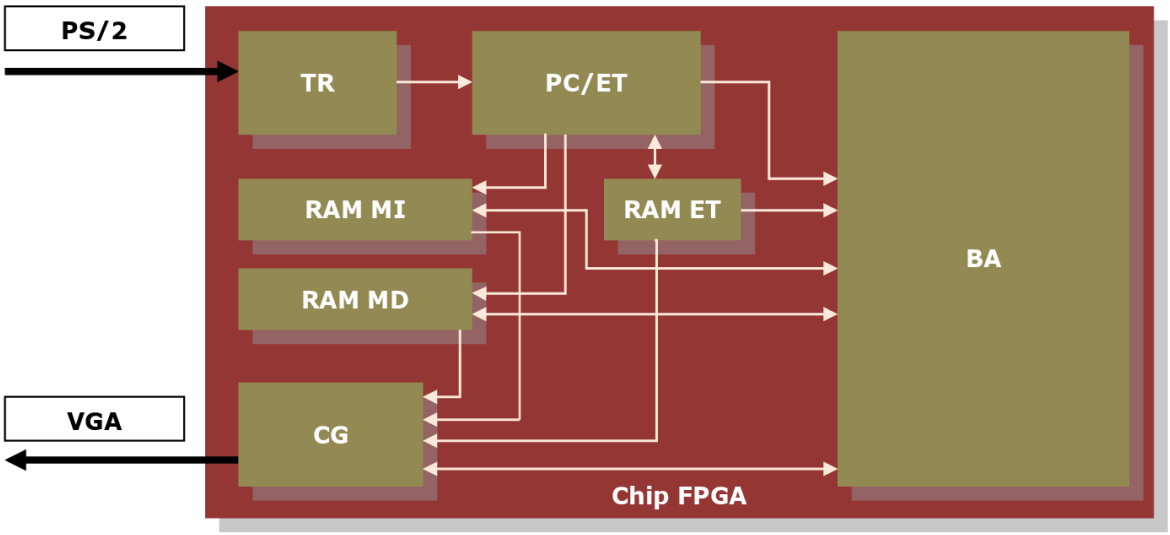
As setas na figura 22, apontam sempre para o bloco que sofre alteração a partir da origem dela. Quando a seta é bidirecional, significa que os dois lados sofrem algum tipo de alteração nos dados internos.

Dentre aos kits de desenvolvimento FPGA existentes, há variações com relação aos recursos eletrônicos presentes nelas. A exemplo disso, pode-se citar memória RAM, que é um dos componentes externos mais complexos, devido à necessidade de sincronizar o circuito com ela, e ainda conseguir executar comandos dentro do tempo de resposta dela. Para isso, desenvolveu-se uma arquitetura que permita ser modular para todos os componentes citados na lista.

Para que a modularização seja feita, foram criados padrões de comunicação entre os componentes, de modo que permite-se desenvolver novos controladores, ou até novas arquiteturas (diferentes da MIPS) para uso no sistema. Isso explica a forma como foi planejado na figura 22.

Dos blocos supracitados, é importante relatar que, caso não se consiga desenvolver e testar o controlador de memória para comunicação com a memória RAM, eles podem facilmente ser substituídos por blocos de memória *on-chip*,que não causa impacto na arquitetura do projeto, devido a sua modularização e também pelo fato do próprio compilador da linguagem gerar código para controle e gerenciamento da memória. Com essa alteração, a arquitetura ficará da forma que está exibida na figura **??**.

Figura 23 – Arquitetura proposta para o desenvolvimento do ambiente em FPGA utilizando memória *on-chip*.



(Fonte: O Autor)

Na figura 23, observa-se, caso a implementação use os blocos memória on-chip ao invés da memória interna, a substituição dos blocos CM e GM pelos seguintes blocos:

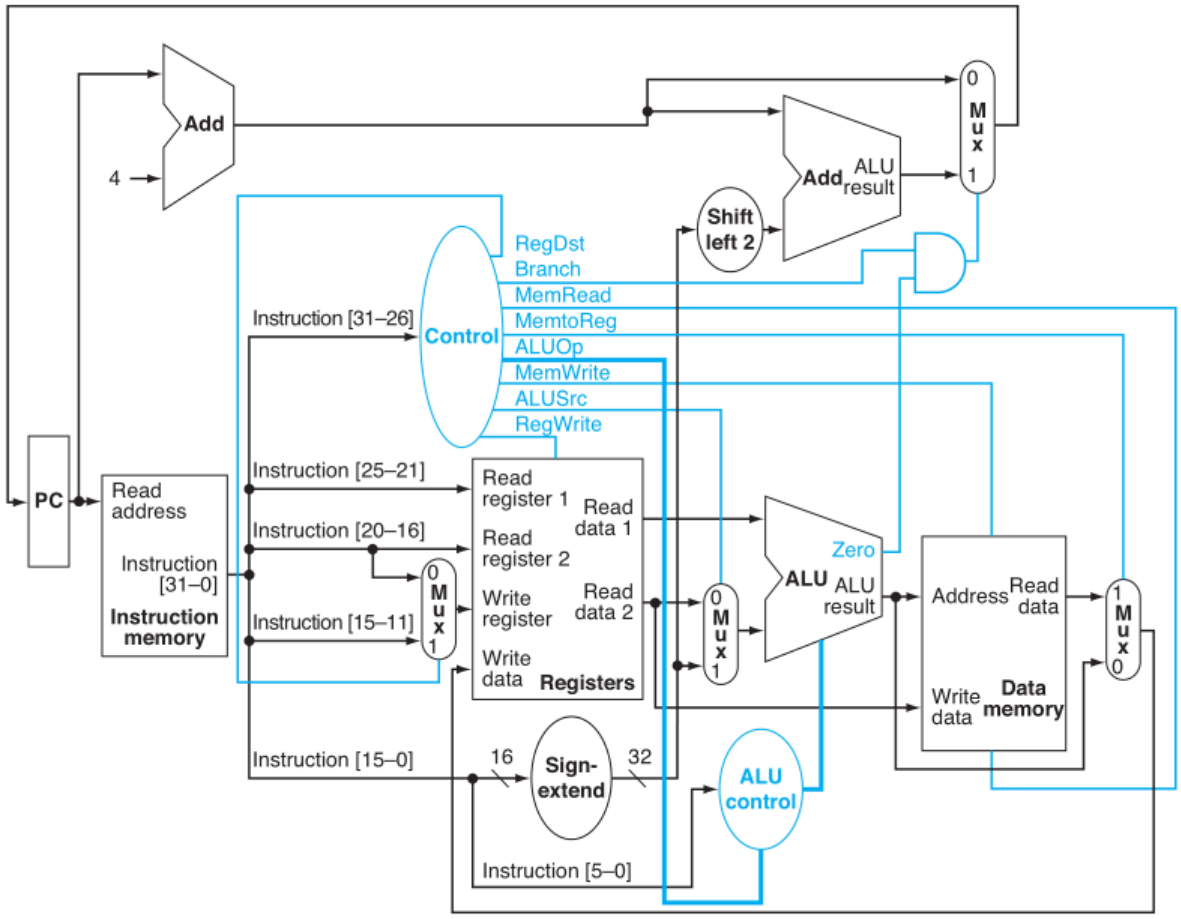
- RAM MI (RAM de Memória de Instruções): Um bloco responsável apenas para armazenar as instruções para a arquitetura;
- RAM MD (RAM de Memória de Dados): Bloco responsável por armazenar dados da arquitetura através das instruções sw e lw;
- e, RAM ET (RAM do Editor de Texto): Memória responsável para guardar uma quantidade ínfima de dados para traduzir para as instruções através de um montador.

As setas da figura 23, seguem o mesmo padrão da figura 22.

4.1 Arquitetura MIPS Monociclo

A arquitetura é um dos pontos centrais deste trabalho, onde é feita a busca de informações internamente para exibição, revelando a forma que as instruções são executadas, além de dados presentes nas memória ROM e RAM e também no Banco de Registradores. O caminho de dados utilizado para implementar a arquitetura se encontra na figura 24.

Figura 24 – Caminho de dados da arquitetura MIPS.



(PATTERSON; HENNESSY, 2012)

A implementação feita da arquitetura MIPS atende à descrição feita por Hennessy e Patterson (2008), em seu livro, em sua forma monocíclica, com o diferencial de uma instrução: addi. As instruções que a arquitetura implementada permite executar estão presentes na tabela 9.

Algumas adequações foram realizadas no projeto com relação às memórias RAM e ROM, pois a sintetização destes tipos de componentes utiliza muitas LE. Como solução, a memória SDRAM utilizada foi dividida, simbolicamente, em três parte:

- 1 x banco de memória para armazenamento de instruções, provendo a capacidade de armazenamento de 256 instruções de 32 bits;
- 1 x banco de memória para armazenamento de dados, com capacidade de 256 palavras de 32 bits;
- e, 2 bancos de memória para uso do editor de texto, com capacidade de armazenamento de 2048 palavras de 8 bits.

Tabela 9 – Instruções com suas formas de uso e funcionamento.

Instrução	Uso	Ação
add	add $1, $2, $3	$1 = $2 + $3
addi	addi $1, $2, 3	$1 = $2 + 3
sub	sub $1, $2, $3	$1 = $2 - $3
and	and $1, $2, $3	$1 = $2 && $3
or	or $1, $2, $3	$1 = $2 || $3
lw	lw $1, 0($2)	$1=$2[0]
sw	sw $1, 4($2)	$1[1]=$2
beq	beq $1, $2, 0	if($1==$2) goto 0;
j	j 0fffffff	goto 0x0fffffff;

(Fonte: O Autor)

Caso o desenvolvimento do controlador de memória não funcione corretamente, a ideia é reduzir significativamente a quantidade de instruções e dados a serem armazenados. Para isso, é desenvolvido um bloco que se parece com uma memória RAM ou ROM, para que o compilador da linguagem VHDL consiga deduzir que aquele bloco deve usar a memória presente, internamente, no chip. Os valores selecionados para armazenamento foram de:

- 1 x bloco de memória para armazenamento de instruções, com capacidade de 32 instruções de 32 bits;
- 1 x bloco de memória para armazenamento de dados, com capacidade de 32 palavras de 32 bits;
- 1 x bloco de memória para uso do editor de texto, com capacidade de 32 letras de 8 bits;

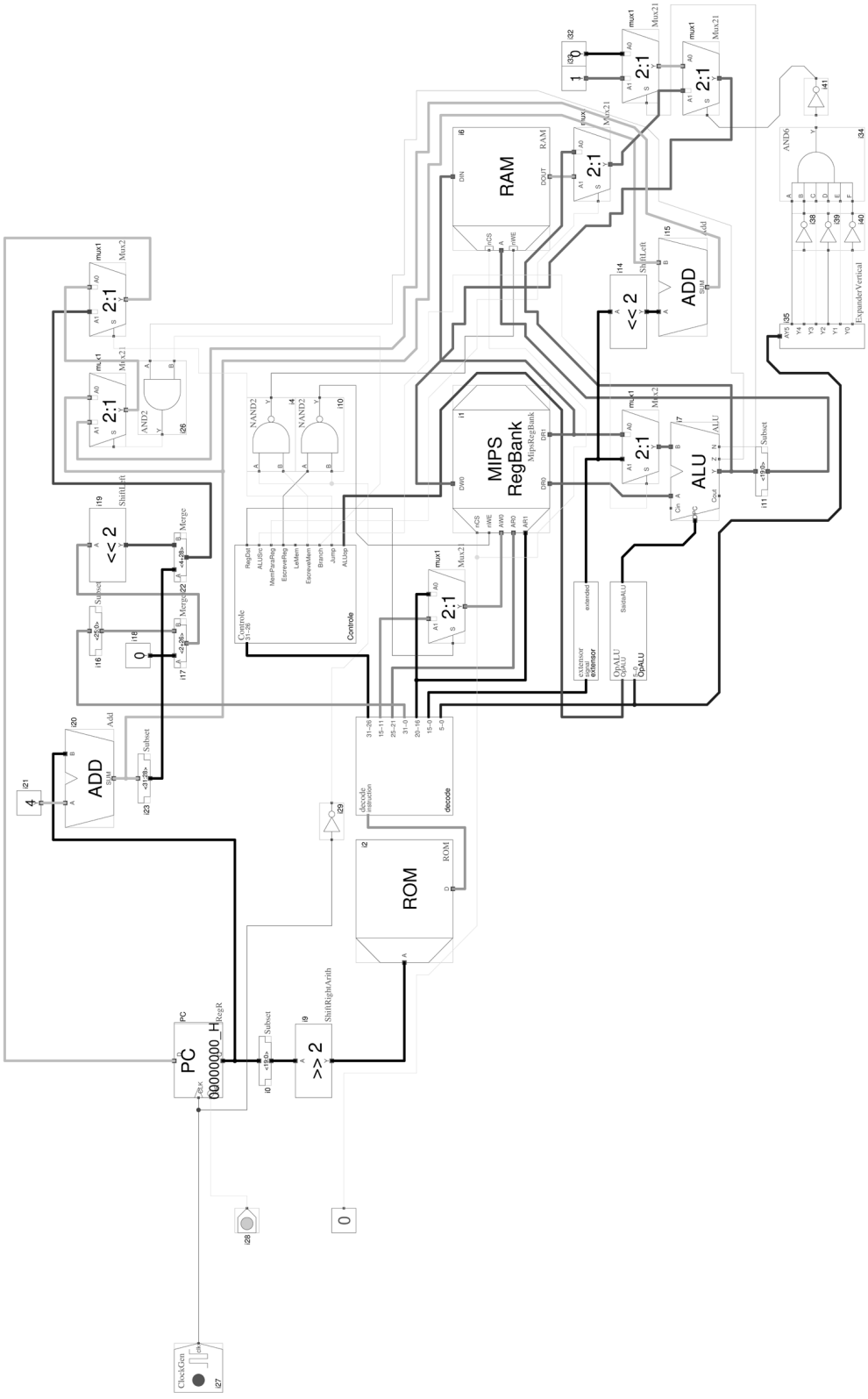
Caso seja sintetizada a memória ROM, por exemplo, dentro do *chip* FPGA, seriam utilizadas 8192 LE, o que ultrapassa a quantidade disponibilizada por uma das placas de desenvolvimento proposta para este projeto, que possui 6272 LE.

Como a memória utilizada possui 16 bits, por palavra, para realizar a leitura das instruções e o armazenamento de dados, será preciso utilizar 2 palavras, o que torna-se possível o armazenamento de instruções de 32 bits. Consequentemente, faz-se necessário a implementação de um gerenciador de memória para solicitar comandos.

Caso a memória utilizada seja a interna, não é necessário utilizar controlador ou gerenciador algum, pois o próprio compilador deduz a utilização dos mesmos e os sintetiza, economizando tempo para o desenvolvedor.

O desenvolvimento deste bloco inicia-se na implementação do circuito através do software Hades, um ambiente de simulação e desenho de circuitos a nível RTL (*Register Transfer Level*). Isto é feito para compreender o comportamento do circuito, frente às instruções, e como os dados percorrem pelo caminho de dados. O desenho do circuito feito encontra-se na figura 25.

Figura 25 – Caminho de dados da arquitetura MIPS desenhado no software Hades.



(Fonte: O Autor)

4.2 Estudo de VHDL e Implementação da Arquitetura MIPS em VHDL

Após a completa implementação no Hades, inicia-se então os estudos da linguagem VHDL, para implementação da arquitetura para o FPGA.

A VHDL, como qualquer outra linguagem, permite facilitar a vida dos desenvolvedores, de forma que a linguagem fica mais próxima da linguagem humana, facilitando o seu entendimento, como foi proposto. No caso do VHDL, ela permite entender como um hardware deve se comportar diante de algumas entradas e saídas.

A linguagem possui um forte pensamento no paralelismo de fluxo de informações, assim como em circuitos eletrônicos, no qual ela é focada. Ela utiliza a organização em entidades (*entity*), como forma de separar componentes, informar quais as entradas e saídas e também auxiliar o compilador a compreender que tipo de componente está sendo sintetizado. Isso permite auxiliar a produção de códigos, para utilizar outros recursos que não sejam os próprios blocos do chip. As entidades devem ser colocadas umas dentro das outras, caso sejam usadas. Para tanto, deve-se utilizar uma entidade-pai, para representar o bloco do circuito inteiro, como é feito na figura 22.

Na figura 25, pode-se observar vários blocos grandes como ROM, RegBank, ALU, dentre outros. A partir desse ponto, é possível pensar que todos estes blocos irão se tornar entidades ao descrever a arquitetura na linguagem.

As conexões entre os blocos são feitas através de sinais (*signals*), os quais devem ter as mesmas características em que serão usados, como barramento e tipo.

Durante a implementação das entidades, é possível testar o seu funcionamento, através de simulações de forma de onda, presente, no software Quartus, o Altera ModelSim. Nela, é possível acompanhar o fluxo de dados passados por dentro da entidade. A cada entrada diferente, é possível visualizar a saída correspondente e, assim, corrigir erros, antes mesmo da programação do projeto no kit de desenvolvimento. Na figura 26, é possível visualizar um dos testes realizados, utilizando a ferramenta para testar a execução da instrução *sub*. Desse modo, é possível observar a execução da subtração do registrador 11 do 10 e armazenando o resultado no registrador 9.

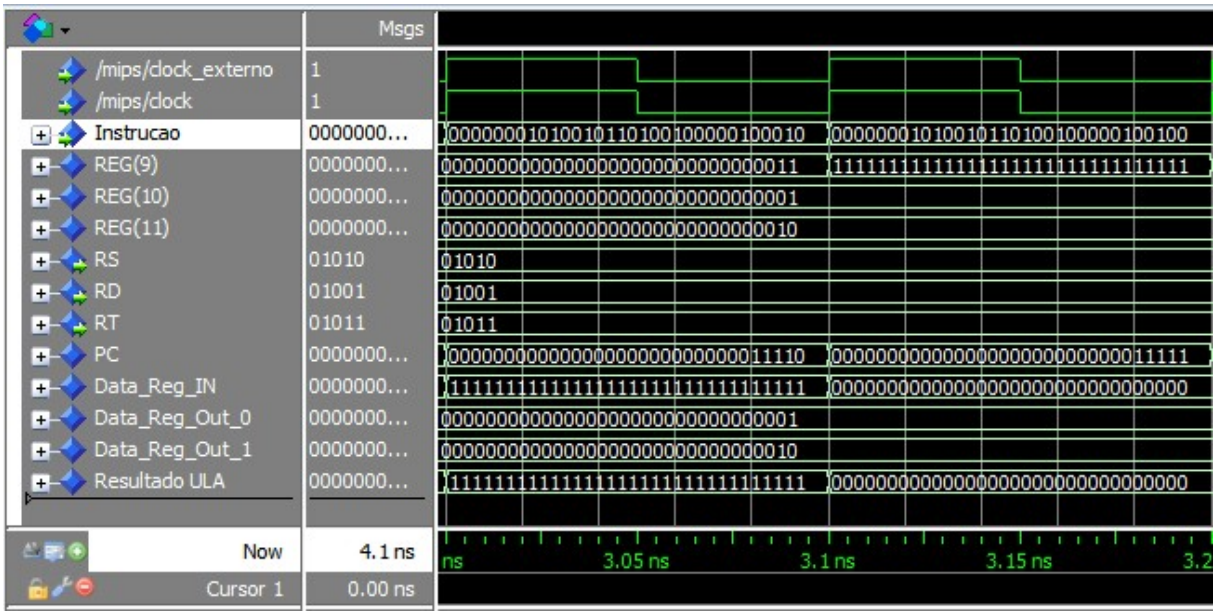
4.3 Controladores

Os controladores têm grande importância neste trabalho, dada a necessidade de receber e enviar informações do chip, externamente e internamente. Para isso, nas subseções seguintes, é descrito o seu desenvolvimento, de modo a esclarecer o seu funcionamento.

4.3.1 Controladora Gráfica

Para a implementação deste controlador, foi utilizada uma técnica proposta por Skliarova (2005) em "Desenvolvimento de circuitos reconfiguráveis que interagem com um monitor VGA". Ele se vale de artifícios para simplificar o desenho da tela, dividindo a resolução em vários blocos maiores, de tamanho 16 *pixels* de altura por 16 *pixels* de largura. Utilizando uma

Figura 26 – Simulação da arquitetura MIPS em formas onda executando a instrução *sub t1 t2 t3*.



(Fonte: O Autor)

memória ROM, com letras armazenadas exatamente do tamanho do bloco, é feita uma busca, linha a linha, de cada bloco para sua exibição no monitor.

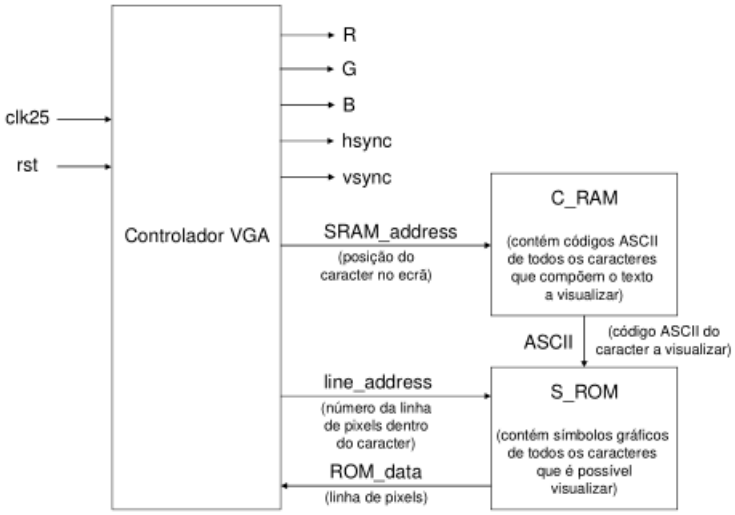
Cada símbolo é armazenado em uma área da memória ROM. O modelo proposto utiliza um símbolo de 16 x 16 *pixels*. Neste trabalho, a placa de desenvolvimento não possui memória ROM externa do *chip* FPGA, sendo necessário sintetizá-la. Tendo isso em vista, a forma utilizada para reduzir o custo no uso de LUTs e blocos de memória é a redução da resolução do símbolo para 8 x 16 *pixels*. Apesar de uma largura inferior, os símbolos não possuem grande distorção em sua exibição. Na figura 27, pode-se observar a arquitetura proposta por Skliarova (2005) para a exibição de imagens no modo texto.

Um diferencial para este trabalho, comparando-o com Skliarova (2005), é que o autor utiliza memória RAM para armazenamento dos blocos de caracteres a serem exibidos, enquanto que, aqui, isso é impraticável, devido à limitação dos dispositivos com seus blocos de memória. Para resolver este impasse, pode-se utilizar duas soluções: memória RAM externa ou um conjunto de blocos condicionais internos, que retornem os mesmos resultados que uma RAM externa. Para essas duas soluções, temos as seguintes vantagem e desvantagens:

- Memória RAM Externa:
  - Vantagem: maior maleabilidade dos componentes, permitindo alterar a informação, em qualquer posição na tela, visto que todas as áreas da memória estariam disponíveis para alteração.
  - Desvantagem: exige um controle de informações muito grande, para cada posição do que deve ou não ser alterado. Outro fator relevante é que o uso de memória RAM

Figura 27 – Estrutura de um simples controlador VGA no modo de texto.

```
65 => ("0000011000000000",  --A
       "0000111100000000",
       "0001111110000000",
       "0011100111000000",
       "0011100111000000",
       "0011100111000000",
       "0011100111000000",
       "0111111111100000",
       "0111111111100000",
       "0111000001110000",
       "1110000000111000",
       "1110000000111000",
       "0000000000000000",
       "0000000000000000",
       "0000000000000000",
       "0000000000000000")
```



(SKLIAROVA, 2005, p.629)

externa reduz a quantidade de *words* disponíveis para a memória do editor de texto RAM e ROM para a arquitetura. Por último, a memória RAM não armazena dados, se for desligada, e também não pode ser inicializada sem os dados.

- Blocos de condições:
  - Vantagem: a lógica combinacional reduz o uso de blocos internos drasticamente, através do compilador VHDL, visto que muitos resultados são parecidos. Outro ponto positivo é a maior robustez ao buscar dados, já que a memória RAM externa possui latência ao fazê-lo. Some-se a isso, a vantagem de que a interface existirá, independentemente do dispositivo estar desligado ou não.
  - Desvantagem: todas as partes da tela podem ser alteradas, assim como na memória RAM externa, porém a diversidade de informações é reduzida apenas para um conjunto de dados.

Analisando as vantagens e desvantagens, optou-se por utilizar os blocos condicionais, devido à maior robustez e também porque os dados, a serem exibidos no monitor, serão com menor taxa de variação.

Para o desenho dos blocos, será utilizada, na totalidade, uma memória ROM interna de 4096 palavras, com 8 bits de tamanho para cada palavra. Para chegar a esses valores foram definidas as seguintes proposições:

- Cada posição da memória, refletirá apenas uma linha de cada caractere, ou seja, 8 bits;
- A cada 16 posições, inicia-se um novo caractere;
- Serão disponibilizados espaços para os 256 caracteres da tabela ASCII;

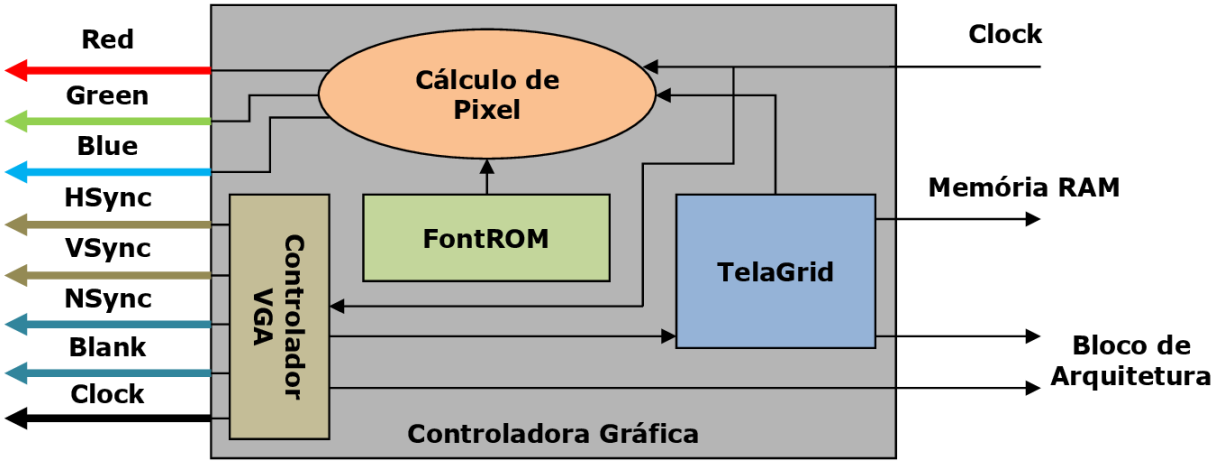
Tendo as proposições acima, pode-se calcular o tamanho da memória a ser utilizada:

$$256 \text{ Caracteres } * 16 \textit{ pixels} \text{ de altura} = 4096 \textit{ posições}$$

A forma de trabalho de um controlador VGA é a de um contador, pois conta a quantidade de pulsos de *clock*, para alterar a saída e, desse modo, informar ao monitor quando se inicia ou termina uma linha ou coluna de *pixels*. Aproveitando desta contagem, é retornada a posição da linha e coluna, em que está sendo desenhada a tela. Também foi pensada a geração de pulsos de *clock*, quando o bloco selecionado é trocado, ou seja, a cada 8 pixels de largura é feito um pulso e, também, a cada 16 pixels de altura.

Esses pulsos e a contagem de posições servem para auxiliar e facilitar a exibição de dados na tela. Desse modo, os pulsos auxiliam, principalmente, para dados dinâmicos e, assim, automatizar o processo, sem armazenar qualquer bit de informação de memória internamente.

Figura 28 – Estrutura da controladora gráfica.



(Fonte: O Autor)

Finalizado o desenvolvimento do bloco do controlador gráfico, temos a figura 28. Nela, é possível visualizar os blocos implementados internamente,onde temos o seguinte:

- Controlador VGA: Área responsável por controlar os sinais de sincronização com o monitor e a geração de sinais de posições para uso interno;
- FontROM: Local de armazenamento das letras desenhadas em binário;
- TelaGrid: Bloco funcional responsável por retornar os blocos, com suas cores, a serem desenhados de acordo com as coordenadas;
- Cálculo de pixel: Processo que ocorre para seleção do pixel na FontROM, de acordo com a posição informada pelo controlador VGA e o caractere informado pela TelaGrid.

No bloco TelaGrid, foi criado um padrão de 11 bits, dos quais os 8 menos significativos representam um código ASCII e os 3 bits mais significativos representam as cores Vermelho, Verde e Azul, respectivamente do mais significativo para o menos significativo. Como é desenhado de forma binária os blocos, os locais que possuem valor '0' representam o fundo (*background*) e os de valor '1' representam o caractere na frente do fundo (*foreground*), assim torna-se possível visualizar os caracteres na tela.

As placas de desenvolvimento utilizadas possuem diferenças na forma de exibição de imagens por meio da interface VGA. A EasyFPGA 2.2a possui comunicação direta com a interface, utiliza apenas 1 bit por cor e, como a tensão de comunicação do chip é de 3.3V, as cores tornam-se um pouco mais escuras, com tons acinzentados. Já a placa Altera DE2-35C possui um conversor de sinais digitais para analógicos de 3 canais, com barramentos de 10 bits por canal, ou seja a tonalidade de cada cor pode variar entre 0 e 1023 tons, o que permite uma grande variedade de cores. Como inicialmente será utilizado a placa EasyFPGA 2.2a, caso se torne-necessária a utilização da Altera DE2-35C, os sinais de cores de 3 bits, irão apenas para os bits mais significativos de cada canal, respectivamente com sua cor, pois assim é simplificado o desenvolvimento desta etapa.

Conforme citado no capítulo 2, é preciso gerar um sinal de *clock* que alcance à frequência necessária para a exibição da imagem em determinada resolução e taxa de atualização. Como o monitor a ser utilizado possui resolução de 1360x768, temos, de acordo com a VESA, os seguintes valores de sincronização:

Tendo as informações da tabela e utilizando a equação explicada no capítulo 2.1.3, tem-se o resultado de 85,478 Mhz, algo próximo à frequência informada pela própria VESA, que é de 85,500Mhz. Segundo o documento "VESA MONITOR TIMING STANDARD", a margem aceita para essa frequência é de 0,5%, o que fornece um intervalo de 85,0725 õ85,9275 Mhz, logo a frequência calculada é ideal.

Como os geradores de *clock*. presentes nas placas, possuem apenas 50MHz, um valor bem abaixo do necessário, torna-se indispensável utilizar o PLL presente no chip FPGA para multiplicá-lo e elevar a sua frequência até esse intervalo. Para isso, o Quartus possui uma fer-

Tabela 10 – Parâmetros detalhados de temporização para a resolução 1360x768 (Adaptado).

Resolução	1360x768
Sincronização Horizontal	
Front Porch	64
Sync Pulse	112
Back Porch	256
Largura da Resolução	1360
Sincronização Vertical	
Front Porch	3
Sync Pulse	6
Back Porch	18
Altura da Resolução	768

(VESA, 2007)

ramenta embutida, cujo acesso se dá através do menu *Tools*, clicando na opção *Megafunction Wizardououou*.

Essa ferramenta permite a criação automática de alguns blocos especiais, aos quais o compilador terá acesso e ativará algum recurso que não use as CLBs, neste caso, o PLL. Durante a configuração deste recurso, são solicitadas algumas informações, porém o essencial é apenas a frequência de entrada e os valores de divisão e multiplicação para alcançar a frequência desejada. Para o monitor, foi inserida a entrada de 50 MHz e os valores de multiplicação e divisão em 12 e 7, respectivamente, atingindo a frequência de 85.714 Mhz, que está dentro do intervalo, e assim é alcançada a frequência desejada.

Aproveitando deste *clock*, é então usado por todo o sistema como forma de mantê-lo inteiramente sincronizado.

4.3.1.1 Software de desenho em blocos

Para desenhar o que será exibido, é necessário montar as condições ou posicionar cada bloco na memória, porém esse é um trabalho árduo e demorado, tornando necessária a implementação de uma ferramenta para acelerar esse processo e gerando códigos em VHDL, que refletem o que será exibido. Como o intuito do software é desenhar, através de blocos, deu-se o nome de BlockPaint, que significa Pintura em Blocos.

Para a implementação deste software, foram levantados alguns requisitos:

1. Gerar código VHDL referente à interface em blocos condicionais e/ou memória;
2. Desenho de interface WYSIWYG (What you see is what you get) que permite visualizar em tempo real como a interface está se parecendo durante o desenho;
3. Automatizar a inserção de texto através do teclado;
4. Demarcar locais que serão dinâmicos e informá-los, no arquivo gerado, para programação posteriormente, pois os valores são alterados com o tempo;

5. Editor de fonte e exportação para VHDL;
6. Gerar código da fonte;
7. Gerar código VHDL de configuração do controlador, para qualquer resolução, informando, também, a frequência necessária.
8. Salvar o projeto.

A linguagem utilizada para desenvolver o software é o Java, devido ao conhecimento sobre esta linguagem ser maior. O ambiente utilizado para desenvolvê-lo é o Netbeans, por possuir interface gráfica para desenho de telas, facilitando e adiantando o processo de criação.

4.3.1.1.1 Janela Principal

Primeiramente, é desenvolvida a tela responsável por desenhar a interface. São utilizadas as bibliotecas padrões do Java para o desenvolvimento da interface.

A janela principal é a área de trabalho, portanto são inseridos os seguintes recursos:

- Botões com funções de desenho livre, desenho de linha, escrita com captura do teclado e desenho de quadrado.
- Lista com caracteres disponibilizados para seleção e permitir ao usuário inserir este determinado caractere.
- Área de criação, listagem e deleção de áreas dinâmicas.

Para a área de desenho, é utilizada uma instância da classe JPanel, em que são inseridos vários JLabels; estes são fragmentos que irão representar cada bloco de caractere. Feito isso, são necessários os seguintes eventos para cada JLabel:

- mouseClicked: Para quando o usuário clicar em inserir um caractere, que esteja selecionado em uma lista;
- mousePressed: Para permitir ao usuário manter pressionado um dos botões e inserir ou apagar o caractere inserido;
- mouseEntered: Para detectar que o mouse entrou na área do JLabel e fazer as ações do mousePressed;

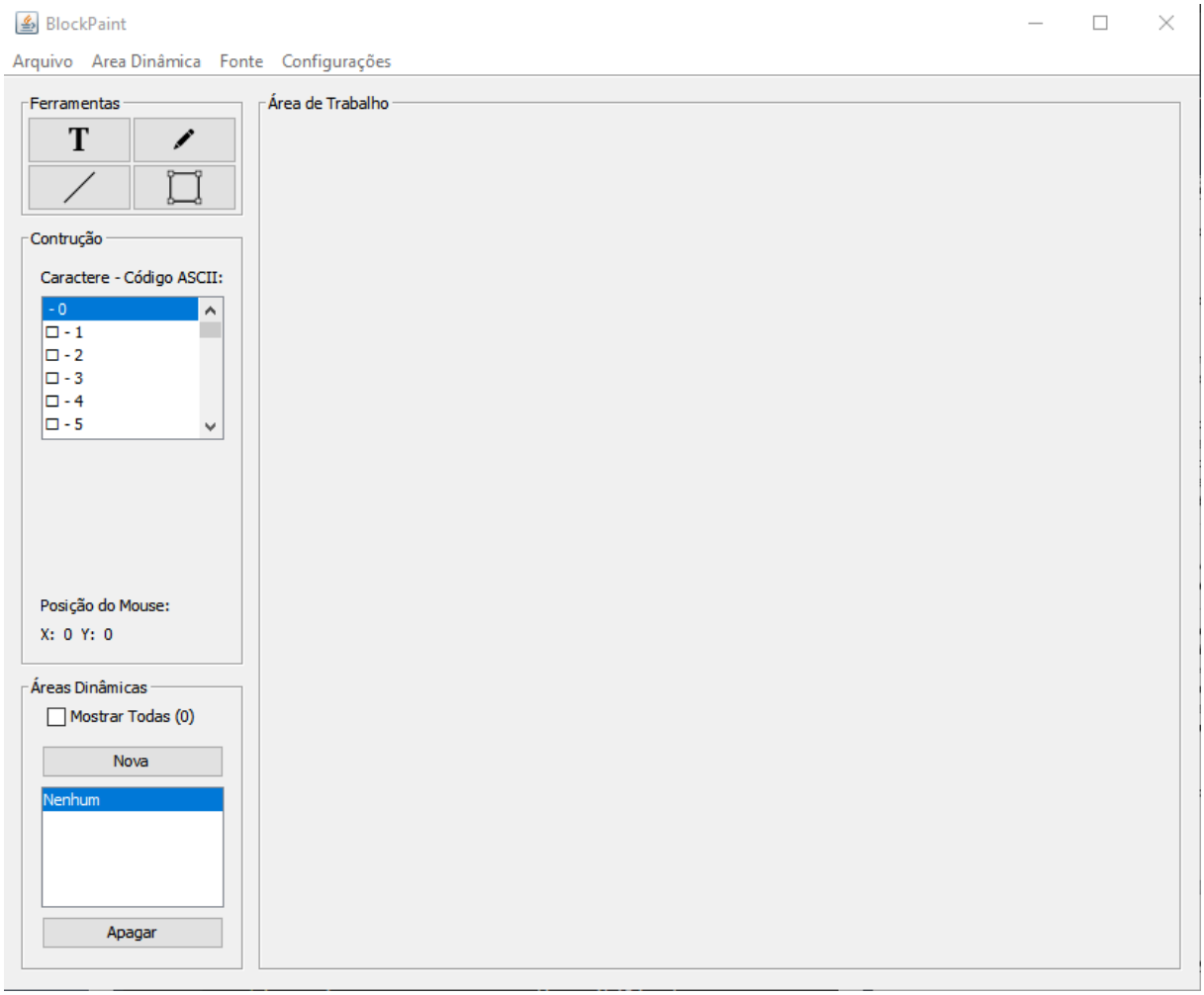
O mouse é o principal meio de desenho. com um *click*, ou mantendo pressionado o botão esquerdo, é inserido o caractere que estiver selecionado na lista. O mesmo ocorre com o botão direito, porém ele sempre apaga.

Para armazenar a grade de caracteres, inserida na área de desenho, é criada uma matriz de JLabel que armazena todos os dados. Uma vez que a capacidade de uso, através dessa classe, é limitada, foi implementada então uma classe derivada da classe JLabel, através de

herança, para que pudesse inserir novos atributos e manter as características da classe pai. Por este meio, o acesso aos dados de uma determinada posição na grade é facilitada.

Com a grade de desenhos finalizada, é necessário então, criar uma forma de armazenar e carregá-la. Desse modo, são desenvolvidas duas funções: uma de leitura e outra de escrita, em que são passadas as dimensões da imagem, dos blocos e a matriz da grade. Assim é feito u armazenamento dos dados em um arquivo com extensão bpp (BlockPaint Project). A imagem da tela produzida pode ser vista na figura 29.

Figura 29 – Tela principal do software BlockPaint.



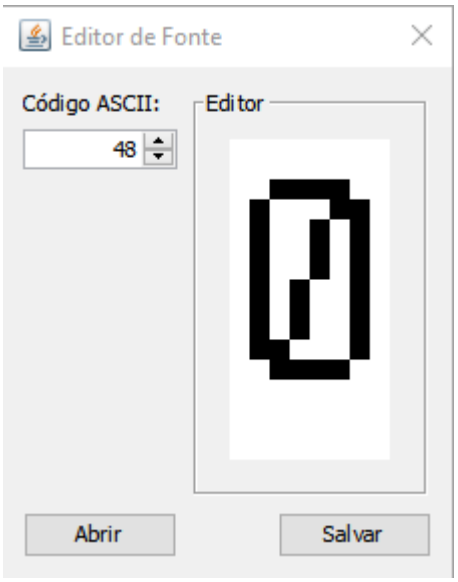
(Fonte: O Autor)

Feito um desenho, para que se possa informar as áreas em que os dados podem ser alterados, foi desenvolvido um meio de selecioná-las e marcar os locais em que serão dinâmicos. Desse modo, é possível auxiliar na geração de códigos. Ao se criar uma área, auxilia-se na geração de código, o que permite separar os locais especificamente para atualizar os valores.

4.3.1.1.2 Desenho de Caracteres

Para o desenho de caracteres, foi criada uma janela específica para esse fim. Ela também tem o mesmo funcionamento da janela de desenho da tela em blocos, cuja a diferença é definida com apenas a seleção do valor em ASCII, a ser desenhado e a inserção dos *pixels* de forma binária, sendo preto '1' e branco '0'. Esta janela pode ser observada na imagem 30.

Figura 30 – Tela de desenho de fontes do software BlockPaint.



(Fonte: O Autor)

A mesma ação de salvar e carregar dados é feita com essa funcionalidade, permitindo carregar automaticamente uma fonte. Ao executar o software BlockPaint, este é carregado e pode iniciar o desenho de interfaces. A extensão utilizada para este feito é bft (Blockpaint Font).

4.3.1.1.3 Geração de Código VHDL

Para a geração de código VHDL, foram criadas duas formas: Geração como Memória RAM e Geração como Bloco de If's. Ao gerar o código, o programa faz a leitura completa da matriz de informações contendo os caracteres inseridos nela e, a partir disso, o código é gerado e salvo com o nome que o usuário desejar.

A geração, como memória RAM, é mais simples de ser elaborada, visto que basta a leitura das colunas de cada linha e armazenar em uma posição de memória. Logo, a cada largura de resolução na memória, é uma linha diferente de blocos. Um exemplo disso pode ser observado através da figura 31.

Já a geração por meio de Blocos de If's é feito por meio de 3 etapas que são as seguintes:

Figura 31 – Exemplo de como é gerado o código de armazenamento em memória.

Grade de caracteres com seus respectivos valores em ASCII

	X1	X2	X3	X4	X5	X6
Y1	1	5	9	13	17	21
Y2	2	6	10	14	18	22
Y3	3	7	11	15	19	23
Y4	4	8	12	16	20	24

Exemplo de como ficará armazenado na memória em linha reta

Posição	Valor	Posição	Valor	Posição	Valor
0	1	8	10	16	19
1	5	9	14	17	23
2	9	10	18	18	4
3	13	11	22	19	8
4	17	12	3	20	12
5	21	13	7	21	16
6	2	14	11	22	20
7	6	15	15	23	24

(Fonte: O Autor)

1. Geração da parte da interface que seja fixa: Como as partes fixas não terão alteração de seu conteúdo, logo não precisa informar quais os locais em que se deve alterar. Então são removidas todas as áreas que são dinâmicas de dentro da matriz da grade de blocos e substituídas por outras células vazias. Essas células são ignoradas na geração de código.

   A fim de que não se tenha um arquivo muito grande e, para auxiliar o compilador na produção de combinações lógicas, são feitas otimizações da seguinte forma:

   a) Gerar retângulos para cada caractere repetido que seja vizinho ao caractere atual, então são armazenadas as posições máximas e mínimas na horizontal e vertical, em um vetor de retângulos;

   b) Analisar os retângulos gerados e unir os máximos e mínimos das colunas e linhas e gerar uma condição ’if’ correspondente;

   c) Depois é feita a organização dos valores que são repetidos, de forma que a condição utilize o operador lógico ’ou’ para unir as condições reduzindo a quantidade de blocos ’if’ para cada uma das áreas.

   d) Armazenar as condições geradas para unir às condições de células dinâmicas.

2. Geração da parte da interface que seja dinâmica: Para as partes dinâmicas são criadas réplicas de matrizes do mesmo tamanho da matriz principal e posicionadas todas as células de apenas uma área dinâmica e, então, é gerado o código para essa área. Esse processo é repetido até que sejam gerado os códigos de todas as áreas dinâmicas.

É feito o mesmo processo de otimização para a geração do código de células dinâmicas, porém separada por áreas.

3. Os códigos de áreas dinâmicas e estáticas são unidas e, então, é salvo o arquivo com o código.

A forma como é otimizada a geração de código pode ser observada na figura 33.

Figura 32 – Exemplo de como é gerado o código de blocos de 'if's.

Grade de caracteres com seus respectivos valores em ASCII

	X1	X2	X3	X4	X5	X6
Y1	1	1	0	3	0	5
Y2	0	1	0	3	4	4
Y3	2	2	3	3	4	4
Y4	2	2	2	3	0	5

Exemplo de como separado os retângulos

<table>
<tr><th></th><th>X1</th><th>X2</th><th>X3</th><th>X4</th><th>X5</th><th>X6</th></tr>
<tr><td>Y1</td><td colspan="2">1</td><td rowspan="2">0</td><td rowspan="4">3</td><td>0</td><td>5</td></tr>
<tr><td>Y2</td><td>0</td><td>1</td><td colspan="2">4</td></tr>
<tr><td>Y3</td><td colspan="2" rowspan="2">2</td><td>3</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Y4</td><td>2</td><td>0</td><td>5</td></tr>
</table>

Algumas condições geradas

```
if (coluna >= 1 and coluna <= 2 and linha >= 1 and linha <= 1) or
(coluna >= 2 and coluna <= 2 and linha >= 2 and linha <= 2) then
      Saída <= 1;
End if;
```

(Fonte: O Autor)

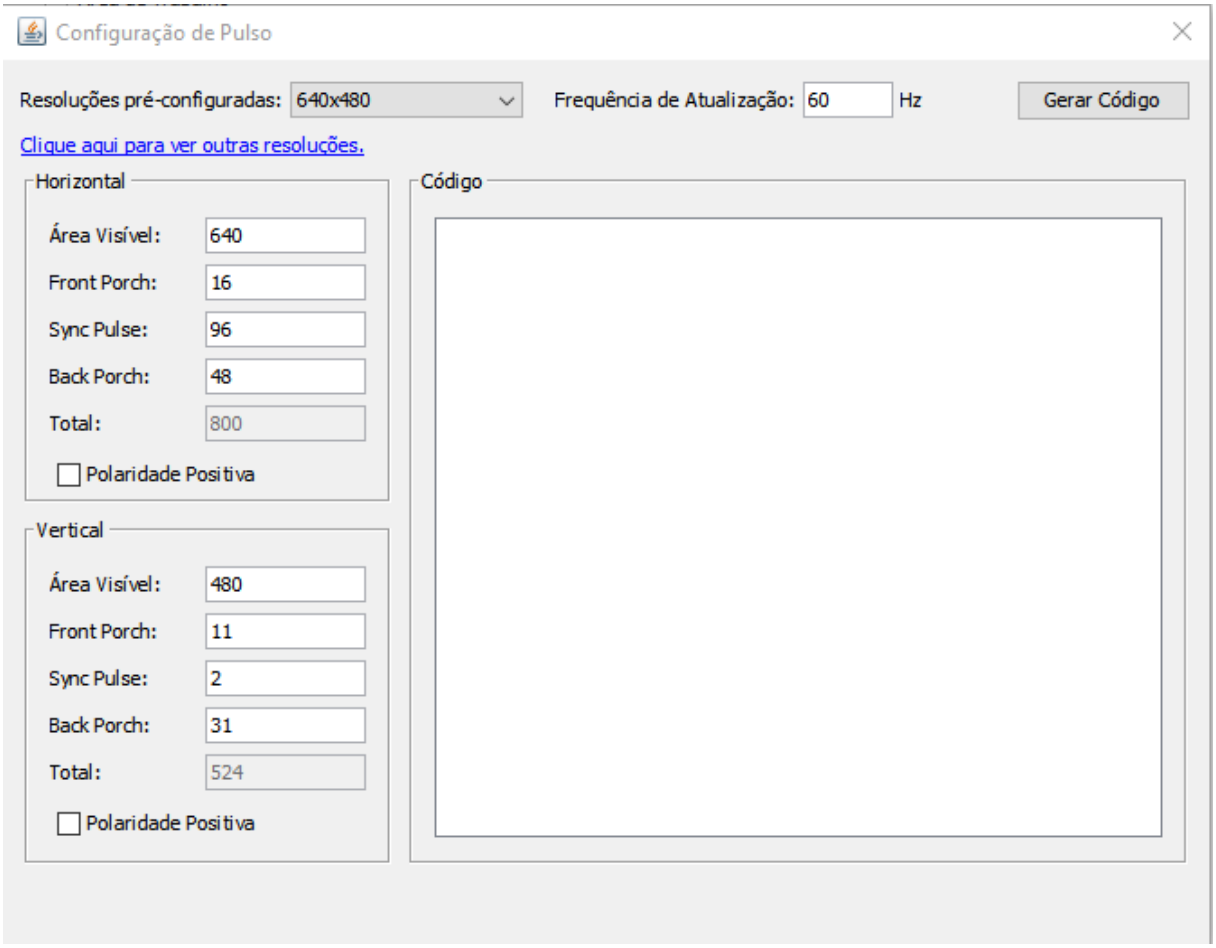
4.3.1.1.4 Geração de código de configuração do controlador.

A geração de código para o controlador é feita através de uma janela em que são informados os parâmetros, com base no que é informado no documento "VESA MONITOR TIMING STANDARD", disponibilizado pela VESA (2007).

A tela basicamente converte os valores na base decimal para a base binária e gera a área do código, em que estão presente as constantes a serem utilizadas pelo controlador e, assim, facilitar a alteração de resolução.

Foram inseridas algumas resoluções pré-definidas, de modo a acelerar os testes da ferramenta para as resoluções diferenciadas, são elas:

Figura 33 – Tela em que são gerados os códigos de configuração do controlador.



(Fonte: O Autor)

- 640x480
- 800x600
- 1280x720
- 1360x768
- 1920x1080

Também foi inserido um link que direciona para o documento que possui todas as resoluções padronizadas pela VESA.

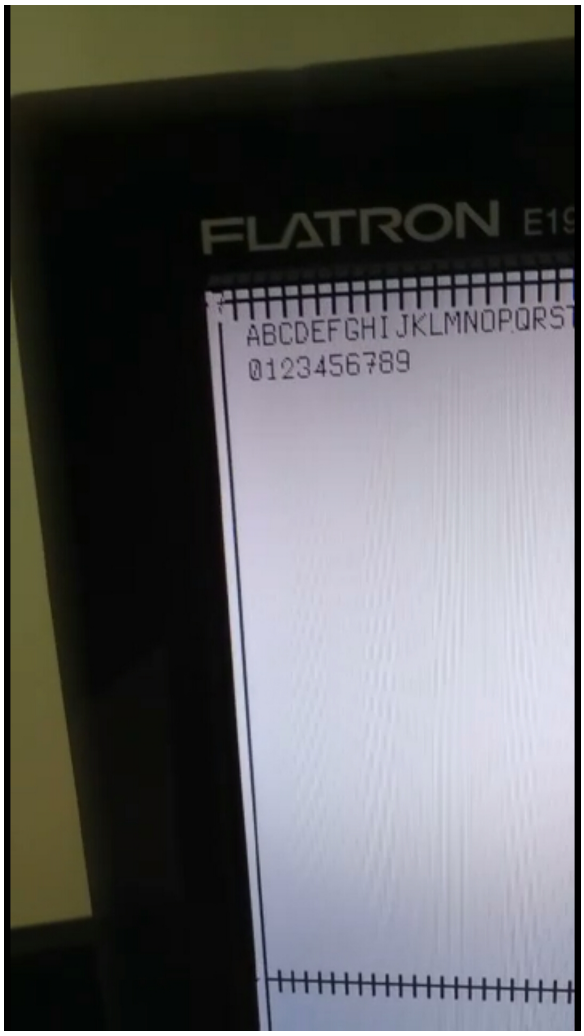
4.3.1.1.5 Finalização

Após finalizada a programação de todo o software BlockPaint, inicia-se o desenho dos caracteres para geração do código e testes em FPGA. Dos caracteres desenhados, tem se o seguinte:

- Letras maiúsculas e minúsculas: A-Z e a-z;
- Números: 0-9;
- Símbolos: $#-*,. │ ┤ ┐ └ ┴ ┬ ├ ─ ┼ ┘ ┌

Após desenhado, foi gerado o código e inserido no projeto para teste do controlador. Também foi gerado o código da configuração do monitor para resolução 1360x768 e gerado um pequeno código para exibição e teste do controlador em FPGA. O resultado pode ser observado na figura 34.

Figura 34 – Tela de teste exibindo os caracteres desenhados no software.



(Fonte: O Autor)

4.3.2 Controlador PS/2

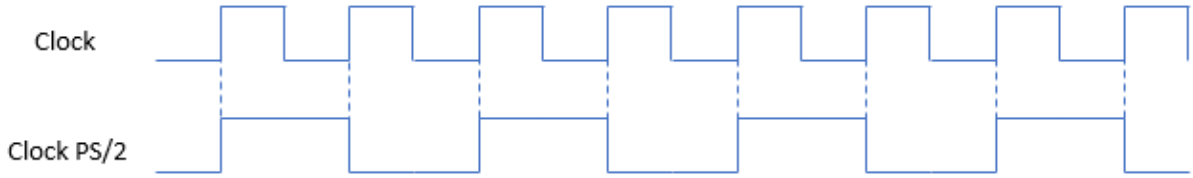
Os dados provenientes do teclado têm o formato do protocolo PS/2 e, com isso, torna-se necessário fazer a conversão dos dados para algum formato conhecido. Para isso um tradutor foi desenvolvido; trata-se de uma máquina de estados que é controlada com base nos *bytes*, recebidos do teclado.

A comunicação, através de uma entrada PS/2, é sempre serial e através de dois pinos principais, sendo um pino de dados, pelo qual a informação será enviada, e um pino de *clock*, por onde é feita a sincronização da informação.

Para receber os dados, é preciso estar em sincronia com o *clock*. A cada subida de *clock* um outro bit estará presente na porta de dados. Como os dados são formados por palavras de 8 bits, é preciso armazenar esta informação até completar a quantidade de bits necessária e, então, tratar os sinais de controle, como os bits de paridade, de inicio e fim. Com isso, tem-se 11 bits por dado enviado.

Um teclado que se comunica, através desta porta, possui um *clock* interno com frequência entre, aproximadamente, 10 e 15 KHz. Como o *clock* da placa de desenvolvimento é mais rápido (frequência de 50 MHz), torna-se possível detectar borda de subida e descida do *clock* do teclado, ou seja, a troca de estado de '0' para '1' e vice-versa. Um circuito simples (com poucas condições e uso de componentes) pode ser feito para coletar os dados através dessa detecção.

Figura 35 – Exemplo de detecção de borda de *clock*.



(Fonte: O Autor)

Este tradutor possui algumas limitações propositais de forma a atender somente alguns caracteres e comandos, simplificando sua implementação. As teclas e comandos os quais ele atende são os seguintes:

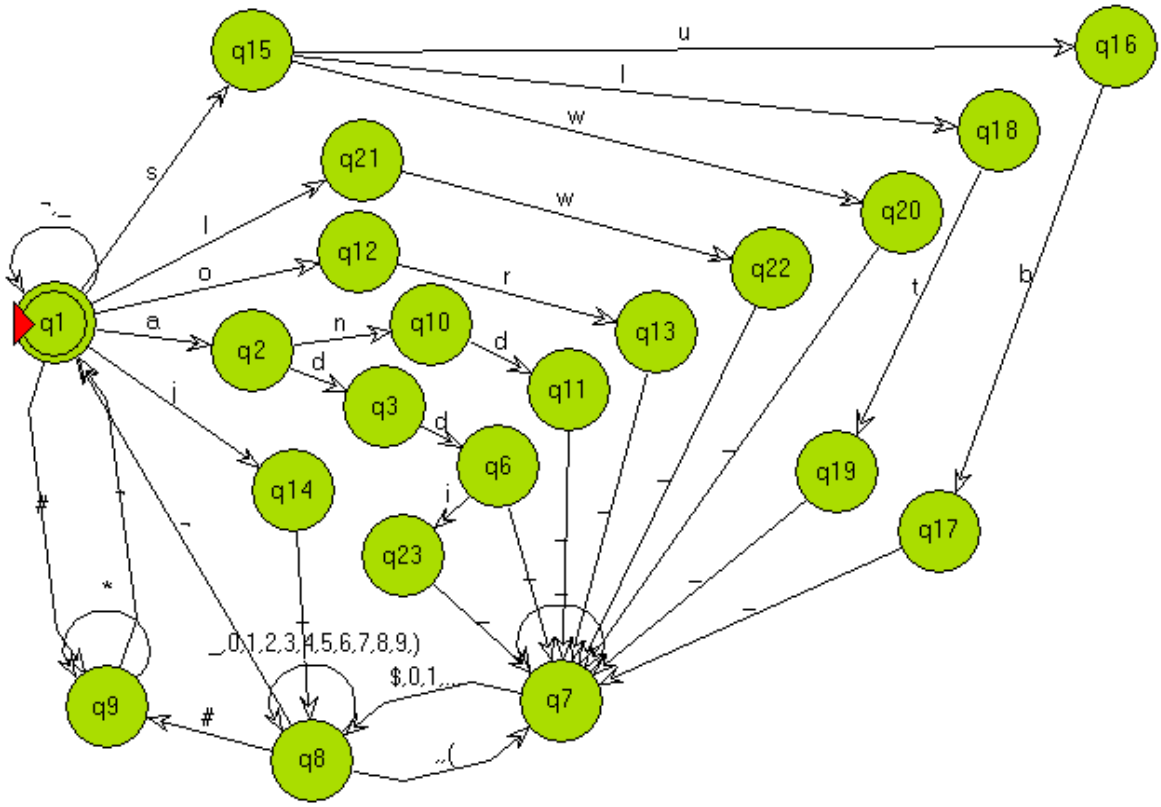
- letras maiúsculas e minúsculas;
- números;
- *capslock* e *shift*;
- espaço e os caracteres especiais '#' e '$';
- e, teclas de funções de F1 até F8.

Esses caracteres são essenciais para que o projeto desenvolvido consiga exercer suas devidas funções. O tratamento dos comandos são feitos por um circuito de processamento de instruções, o qual irá ativar ou desativar outros circuitos, como o recebimento de textos para o editor de texto e o controle da arquitetura, permitindo compilar e executar os programas desenvolvidos no editor de texto.

4.3.3 Montador

O montador possui uma estrutura interna de forma a tratar os caracteres recebidos a partir da memória RAM para montagem do código. Ele é uma máquina de estados que, a partir da leitura de alguns caracteres, consegue ter como resposta uma instrução ou algum erro que consiga encontrar dentro do código digitado pelo usuário.

Figura 36 – Autômato finito definido para a montagem do código de acordo com as instruções definidas.



(Fonte: O Autor)

Na figura 36, podemos observar a máquina de estados planejada. É importante ressaltar que o símbolo '_' (traço-baixo, ou popularmente conhecido, do inglês, *underline*), está representando o espaço do valor 0x20 da tabela ASCII, que é o espaço em branco.

A cada estado é feita a análise da letra recebida e qual o próximo passo que deve ser seguido para finalizar a montagem. Ele consegue converter números em ASCII para binários, para o caso de inserção de instruções do tipo-I e também Tipo-J, que possuem um valor numérico imediato.

É preciso estar atento em relação ao planejamento deste autômato, visto que, por ser longo e toda a leitura ser binária, a sincronia com a memória RAM deve ser realizada adequadamente, de modo a tornar a leitura mais correta, garantindo a melhor performance.

Esta implementação possui algumas limitações em relação às outras soluções propostas por outros trabalhos. Dentre os recursos aceitos, tem-se o seguinte:

- Compreender apenas as instruções: add, addi, sub, and, or, slt, beq, j, lw e sw;
- Comentários iniciados com o caractere '#' e terminam ao final da linha (quebra de linha);
- A referência a um registrador começa com $ e o número do registrar na frente (Exemplo: $0,$2,$3,$4);
- Erros de sintaxe (Exemplo: Instrução não suportada) e estruturação (Exemplo: faltar um registrador em uma instrução que necessite de três).

Pelas características citadas acima, a ideia é utilizar este autômato em uma memória grande que armazene o código do editor de texto. Por apenas alterar seus estados através de entradas, é possível ser utilizado em qualquer tamanho de código, desde que seguidas as especificações.

Este montador consegue informar o erro o qual o código está com problema, desde a conversão de um número para binário, até o estado em que o caractere está incorreto.

4.4 Controladora de Memória SDRAM

Uma memória SDRAM possui sincronização complexa para ser manipulada diretamente por um circuito que a utilize. Para isso, é desenvolvido um controlador que permite realizar o interfaceamento entre essas duas partes.

Um controlador de memória deve estar atento aos tempos com que a memória trabalha. Alguns dos tempos de latência que a memória utiliza neste trabalho podem ser encontrados na tabela 11. Nela, há os principais atrasos, que são levados em consideração para a construção da controladora.

Tabela 11 – AC CHARACTERISTICS II.

Parameter	Symbol	Min	Max	Unit
$\overline{RAS}$ Cycle Time (Operation)	tRC	63	-	ns
$\overline{RAS}$ Cycle Time (Auto Refresh)	tRRC	63	-	ns
$\overline{RAS}$ to $\overline{CAS}$ Delay	tRCD	20	-	ns
$\overline{RAS}$ Active Time	tRAS	42	100K	ns
$\overline{RAS}$ Precharge Time	tRP	20	-	ns
$\overline{RAS}$ to $\overline{RAS}$ Bank Active Delay	tRRD	14	-	ns
$\overline{CAS}$ to $\overline{CAS}$ Delay	tCCD	1	-	CLK
Write Command to Data-In Delay	tWTL	0	-	CLK
Data-in to Precharge Command	tDPL	2	-	CLK
Data-In to Active Command Delay	tDAL	tDPL+tRP	-	CLK
MRS to New Command Delay	tMRD	2	-	CLK
Refresh Time	tREF	-	64	ms

(HYNIX, 2007, p.11)

É possível perceber que a tabela 11 possui referência aos tempos máximo e mínimo de respostas para determinado comando. O tempo mínimo reflete a latência que a memória

possui diante de algum comando. O tempo máximo é a quantidade de tempo em que fica aguardando para a sequência de novo comando. Todos eles são utilizados de forma a sincronizar o sistema com a memória e cada um é utilizado como base no comando que se deseja executar.

A memória SDRAM possui alguns recursos de modo a facilitar o seu uso, mas não serão implementados todos os comandos, pois a ideia é o trabalho com uma controladora simplificada. Para isso, é desejável que se consiga executar os comandos de configuração, leitura, escrita e realizar o *refresh*,de modo a não perder os dados.

A configuração da memória é feita através do comando *Mode Register Set*, conforme informado no capítulo 2. Os valores selecionados para os campos e as motivações para tais, são os seguintes:

- *Write Mode*: Burst Write - Pois torna possível a gravação de múltiplas palavras em apenas um comando, aumentando o desempenho e economizando ciclos de *clock* para execução de outros comandos;
- *CAS Latency*: 2 - Para uma melhor sincronização com leitura dos dados para exibição na saída VGA, por ser de 8 bits, um múltiplo de 2.
- *Burst Type*: Sequencial - Pois os dados necessários serão sempre em sequência.
- *Burst Length*: 2 - Como está sendo trabalho em um ambiente limitado, um *buffer* para armazenar um *burst length* muito grande utiliza muitos blocos lógicos, o que desvia do objetivo de desenvolver algo com a menor quantidade de blocos possíveis.

Essa configuração pode ser refeita a qualquer momento durante o tempo de funcionamento estabilizado da memória (após o tempo de estabilização de $200\mu s$), porém foi definido manter apenas uma configuração, de modo a simplificar a montagem da controladora e também do gerenciamento da memória. Isso torna o circuito mais simples, apesar de mais lento, por precisar fazer muitas leituras para buscar ou gravar duas palavras por vez.

Após implementada e testada a controladora de memória, iniciou-se alguns testes com prototipação para validação de sua execução. Apesar de sua simulação ter sido executado corretamente, a execução em prototipação FPGA não obteve sucesso, pois não foi possível comprovar que a memória estava lendo ou escrevendo.

Houve muitas buscas e testes com controladores de terceiros para avaliação da memória, todavia mesmo assim não houve a confirmação de que os dados foram armazenados e lidos. Com isso, deixou-se a implementação de memória externa como um trabalho futuro e sendo utilizada a segunda opção, conforme citado inicialmente.

4.5 Blocos de Memória RAM

Como o desenvolvimento da memória não obteve sucesso, inicia-se o desenvolvimento por meio de blocos de memória. Para isso, foram definidos 3 blocos de memória:

- RAM MI (Memória de Instruções): Esta memória possui 32 palavras de 32 bits para armazenamento das instruções, a serem usadas pela arquitetura;
- RAM MD (Memória de Dados): Esta memória possui 32 palavras de 32 bits para armazenamento dos dados, a serem usadas pela arquitetura;
- RAM ET (Memória do Editor de Texto): Esta memória possui 32 palavras de 8 bits para armazenamento dos caracteres do texto, a serem usados pelo montador;

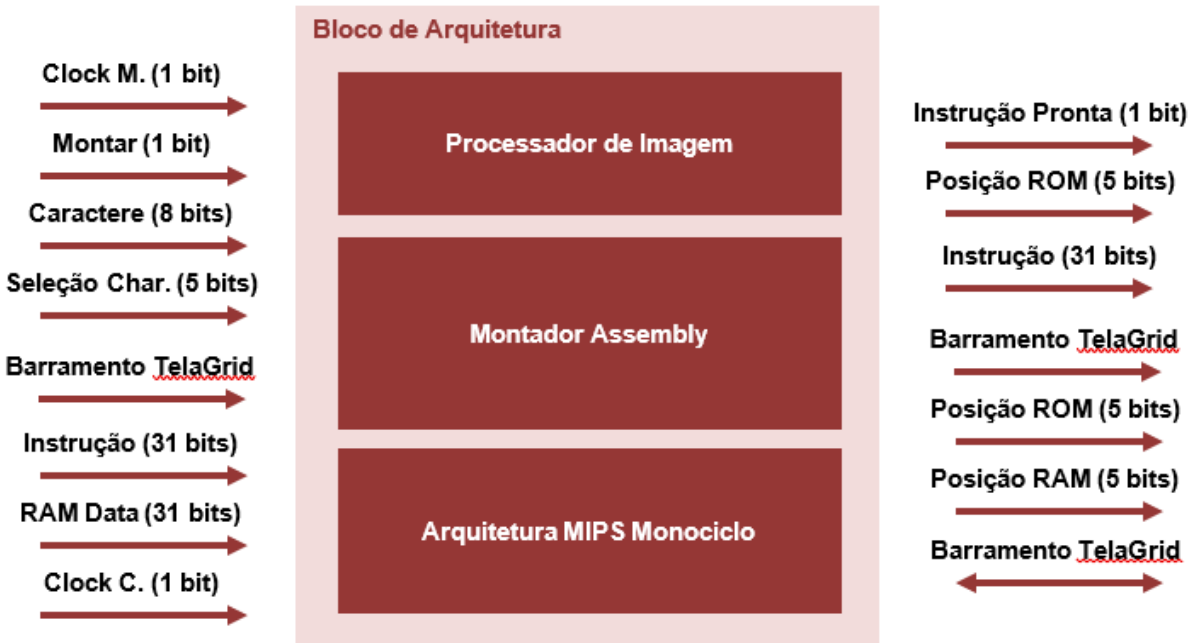
Todas as memórias possuem 3 meios de acesso, sincronizadas por *clock*. Por possuir a semelhança de comportamento de uma memória RAM, então o compilador deduz que deve-se utilizar blocos de memória On-chip e, assim, economizar blocos de memória.

Toda leitura e escrita é feita com apenas um *clock*, fornecendo uma leitura 7 vezes mais rápida que a leitura de dados na memória externa, que possui um atraso de 7 pulsos de *clock*. Logo, o desempenho subiu consideravelmente em detrimento do espaço de armazenamento.

4.6 Bloco de Arquitetura

O bloco de arquitetura é composto por um subconjunto de blocos que tendem a processar os sinais de comandos do usuário, as instruções que a arquitetura suporta, montagem do código de máquina e envio de dados para sinais de vídeo. A interface de comunicação e a subdivisão dos blocos é descrita na figura 37.

Figura 37 – Divisão interna do bloco de arquitetura e suas entradas e saídas.



(Fonte: O Autor)

4.6.1 Processador de Imagem

O processador de imagem é um sub-circuito responsável por processar a imagem a ser exibida do caminho de dados, de acordo com a instrução a ser executada. Para isso, é feita a análise das coordenadas da posição na tela, que é fornecida através do controlador de vídeo, e também a análise da instrução executada.

A imagem a ser exibida é o caminho de dados da arquitetura. A cada instrução executada, as cores das linhas que se comunicam entre os componentes são modificadas, para demonstrar a parte útil da arquitetura para determinada instrução.

Este sub-circuito tem a tendência de se comportar como uma memória, visto que são passadas apenas as informações de posição (coordenadas da tela) e um modificador (instrução) para que se consiga recuperar a informação. Internamente, possui um conjunto de condições e lógicas que são tratadas a partir das entradas.

A única informação a ser retornada é um conjunto de 11 bits, dos quais 8 bits são de um caractere da tabela ASCII e os outros 3 são referentes à cor que será exibida, fornecendo um total de 8 cores possíveis dentro do padrão RGB. Como a cor branca já está sendo utilizada, será usado a amarela para dois valores repetidos.

As cores das linhas serão definidas pelos 3 bits menos significativos do valor que estiver passando pela linha, exceto o "111", pois tornaria a linha invisível por ter a mesma cor do fundo da tela. A quantidade de bits pode variar, conforme o dispositivo FPGA. Como a placa Altera DE2 tem 10 bits para cada cor, esta permite uma variedade alta de cores.

A imagem é desenhada no software implementado, o BlockPaint e, a partir de uma prévia, gerada a partir dele, temos o que é mostrado na figura 38. Esta figura é baseada nas figuras 24 e figura 25. Verifica-se, desta forma, que a aparência da imagem (sua arquitetura) é semelhante à de Patterson e Hennessy (2012).

4.7 Processador de Comandos e Editor de Texto

O processador de comandos é a etapa logo após a tradução dos *scancodes* provenientes do teclado e interpretação deles. Logo, tudo o que é informado pelo usuário através do teclado, é enviado para esse processador. Nele, deve haver um conjunto de comandos processados, que são os seguintes:

- Alternar entre modo de execução e modo de programação.
- No modo Execução:
  - Avançar a execução com um pulso de *clock*;
  - Resetar os registradores, memóra RAM e PC;
- No modo Programação:
  - Selecionar posições de gravação na memória ROM;

  - Selecionar posições no campo de digitar a instrução;
  - Iniciar Montador;
  - Alternar entre o campo de texto e a área de instruções;
  - Apagar instrução selecionada.

Como a memória RAM externa não será usada, o campo de texto usado é limitado a 32 posições, o suficiente para armazenamento de uma instrução inteira e tradução. Então, sempre que executar uma tradução bem sucedida, a área será apagada para receber uma nova instrução.

Para a implementação do processador de comandos, é desenvolvida uma máquina que aguarda a troca de valores de algumas *flags*, isso permite a validação da execução de comandos como a montagem do código *assembly* que precisa de mais de um ciclo para executar, por exemplo.

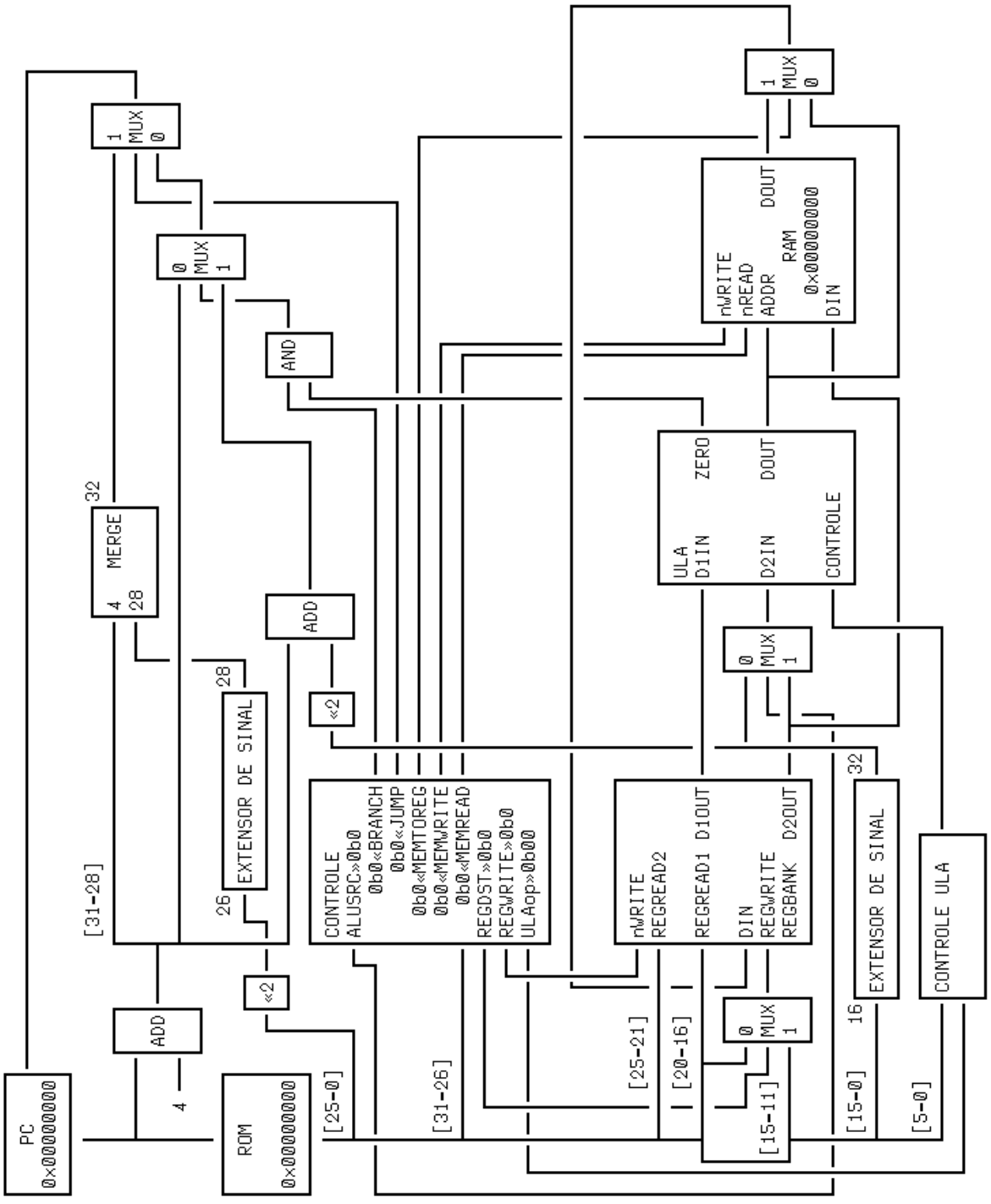
Aproveitando que o processamento é muito rápido, devido ao *clock* ser muito mais rápido que a percepção humana, torna-se dispensável informar ao usuário qualquer operação que esteja ocorrendo, o que se traduz em economia de tempo e lógica para outros componentes.

O processador de comandos é desenvolvido para ter acesso direto à maioria dos componentes presentes no projeto, excluindo apenas à controladora gráfica, que não necessita de interação humana para funcionar.

Os comandos criados foram os seguintes:

- Ctrl+E: Alternar execução e edição;
- Modo Edição:
  - Tab: Alternar entre memória ROM e memória de Edição;
  - Backspace: Apagar um caractere ou um dado na memória ROM;
  - Enter: Iniciar montagem de instrução;
- Modo Execução:
  - Espaço: executar um pulso de *clock* na arquitetura, caso mantenha pressionado a execução será contínua.;
  - Enter: Inicia e finaliza execução contínua de instruções;

Figura 38 – Prévia da imagem a ser exibida da arquitetura produzida pelo software BlockPaint.



(Fonte: O Autor)

5 Resultados

“Nossa maior fraqueza está em desistir.
O caminho mais certo de vencer é tentar mais uma vez.”
Thomas Edison

Este capítulo apresenta os resultados da utilização dos recursos do dispositivo FPGA. Isso foi possível mediante a compilação e sintetização do hardware, a interface gráfica da arquitetura MIPS, com o caminho ativo de cada instrução executada pelas unidades funcionais e o programa de teste para analisar seu comportamento na interface gráfica.

5.1 Análise de dados de compilação.

Ao finalizar a compilação do código, o compilador da linguagem produz vários relatórios para informar ao desenvolvedor a situação do projeto, como uso de recursos, possíveis problemas e análise de desempenho, dentre outras informações. Para a presente pesquisa, o enfoque está no uso de recursos, por isso será analisado.

A tabela 12 informa o uso de blocos de memória e blocos lógicos. Estes últimos podem ser utilizados tanto para lógica combinacional quanto para registradores, sendo, então, dividida a tabela em duas colunas para os dois tipos.

Ao analisar os dados, é possível perceber que a maior parte dos blocos utilizados foram apenas para resolução de problemas de lógica combinacional, com um total de 7782 LEs. Isso se deve ao fato de se recorrer a muitos recursos para resolver problemas de exibição e também a montagem de instruções com o montador.

Percebe-se que o banco de registradores utilizou 2069 blocos e nenhum bloco de memória, o que pode ser melhorado futuramente, com o objetivo economizar blocos lógicos para outras funções. Esse foi o componente que mais consumiu LEs. em todo o projeto, podendo ser um ponto de foco de atenções para implementação de novos projetos utilizando a arquitetura MIPS.

O bloco lógico que utilizou a maior quantidade de blocos de memória foi o que contém as letras desenhadas no software BlockPaint, consumindo 32768 bits de memória. Isso permitiu economizar blocos para lógica combinacional, de modo a comportar todas essas informações e, assim, exibir, com qualidade, todos os dados. Também é possível perceber que alguns bits de memória foram utilizados em outras ocasiões, como nos blocos lógicos que representam a memória ROM e RAM da arquitetura MIPS.

As informações desta tabela representam a utilização, sem otimização, de uso dos recursos. Durante a compilação, é feita a otimização e, posteriormente, apresentado o Relatório de utilização de recursos. 13.

Tabela 12 – Relatório de compilação do uso de recursos do Quartus.

Hierarquia de Compilação de Nós	**Função Combi.**	**Registradores**	**Memória**
|Principal	7782 (1)	2706 (0)	35392
|Bloco_Arquitetura:Bloco_Arquitetura1|	5192 (0)	1372 (0)	128
|Desenho_Monociclo:Desenho|	1976 (1976)	0 (0)	128
|Hexa_to_ASCII:HexaASCII|	0 (0)	0 (0)	128
|altsyncram:altsyncram_component|	0 (0)	0 (0)	128
|altsyncram_avr3:auto_generated|	0 (0)	0 (0)	128
|MIPS:ArquiteturaMIPS|	2638 (161)	1152 (0)	0
|Add:ADD1|	32 (32)	0 (0)	0
|Add:ADD2|	32 (32)	0 (0)	0
|BancoDeRegistradores|	2069 (2069)	1120 (1120)	0
|Controle:CONTROLE1|	12 (12)	0 (0)	0
|MUX2x1:MUX1|	32 (32)	0 (0)	0
|MUX2x1:MUXDATAREG|	32 (32)	0 (0)	0
|MUX2x1:MUXMUXEXT1|	16 (16)	0 (0)	0
|MUX2x1_5:MUXREG|	30 (30)	0 (0)	0
|PC:PC1|	24 (24)	32 (32)	0
|ULA:ULA1|	190 (190)	0 (0)	0
|UlaControle:CONTROLEULA1|	8 (8)	0 (0)	0
|Montador:Montador_Arquitetura|	578 (578)	220 (220)	0
|PLL:PLL1|	0 (0)	0 (0)	0
|altpll:altpll_component|	0 (0)	0 (0)	0
|Processador_Comandos:ProcComandos|	171 (171)	69 (69)	0
|RAM_Extra:RAM_Arquitetura|	768 (768)	1024 (1024)	0
|RAM_Extra:RAM_EDITOR|	14 (14)	19 (19)	512
|altsyncram:RAM_Extra_rtl_0|	0 (0)	0 (0)	256
|altsyncram_8kc1:auto_generated|	0 (0)	0 (0)	256
|altsyncram:RAM_Extra_rtl_1|	0 (0)	0 (0)	256
|altsyncram_2hg1:auto_generated|	0 (0)	0 (0)	256
|RAM_Extra:ROM_Arquitetura|	33 (33)	31 (31)	1984
|altsyncram:RAM_Extra_rtl_0|	0 (0)	0 (0)	992
|altsyncram_0nc1:auto_generated|	0 (0)	0 (0)	992
|altsyncram:RAM_Extra_rtl_1|	0 (0)	0 (0)	992
|altsyncram_0nc1:auto_generated|	0 (0)	0 (0)	992
|Screen:Screen1|	1234 (45)	103 (3)	32768
|FontROM:FontROM1|	0 (0)	0 (0)	32768
|altsyncram:ROM_rtl_0|	0 (0)	0 (0)	32768
|altsyncram_rt61:auto_generated|	0 (0)	0 (0)	32768
|TelaGrid:GRID|	1085 (1085)	12 (12)	0
|vga_controller:VGA|	104 (104)	88 (88)	0
|tradutor:Teclado|	369 (319)	88 (27)	0
|teclado_ps2:teclado_ps2_0|	50 (22)	61 (35)	0
|delay:delay_ps2_clk|	14 (14)	13 (13)	0
|delay:delay_ps2_data|	14 (14)	13 (13)	0

(Fonte: O Autor)

Tabela 13 – Relatório de utilização de recursos após otimização.

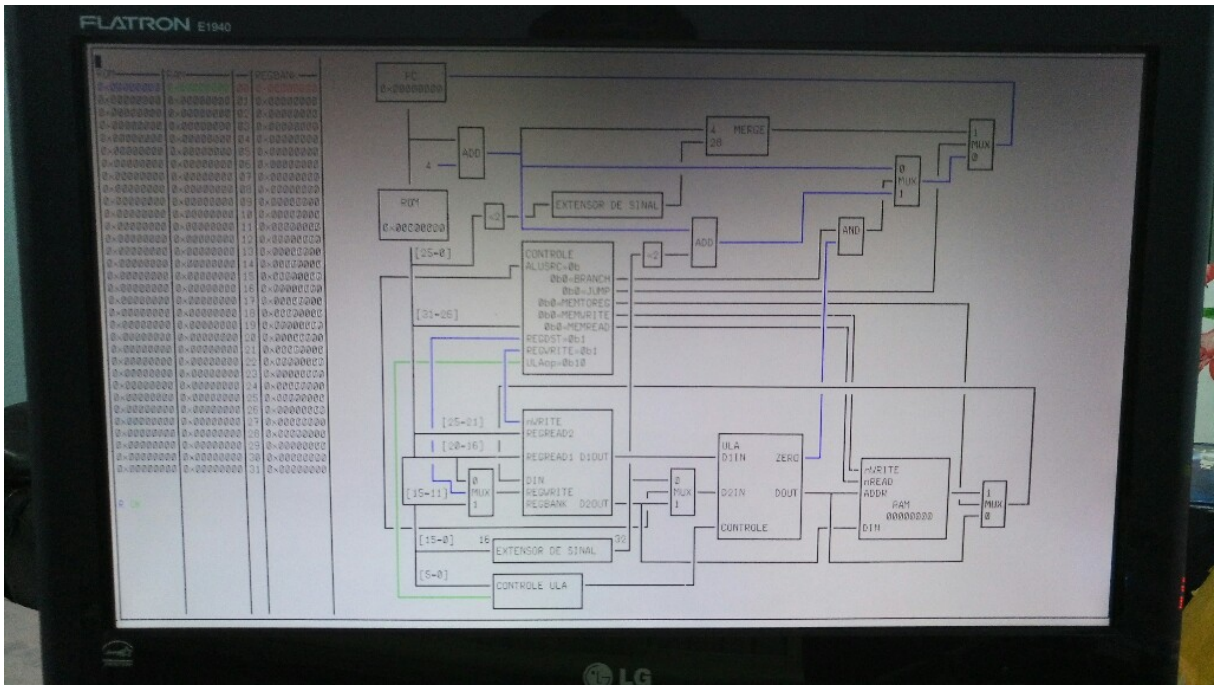
Dispositivo			EP2C35F672C6
Recurso	Utilizado	Total	Proporção
Total de elementos lógicos	9.658	33.216	29%
Total de funções combinacionais	7.787	33.216	23%
Registradores lógicos dedicados	2706	33.216	8%
Pinos	11	475	2%
Bits de Memória	35.392	483.840	7%
PLLs	1	4	25%

(Fonte: O Autor)

5.2 Análise da interface gráfica e interação

Após a sintetização e embarcação da ferramenta, ela pode ser utilizada quantas vezes se desejar, permitindo a programação e exibição da execução das instruções. Na figura 39, é possível visualizar a tela desenvolvida para programação, como também o caminho de dados da arquitetura MIPS, uma área para visualizar os dados da memória RAM, ROM e banco de registradores, além de um campo para escrever as instruções, na extremidade superior esquerda.

Figura 39 – Exibição da tela da ferramenta.



(Fonte: O Autor)

É possível visualizar um cursor para localizar onde será inserida alguma letra ou seleção da posição de memória ou removida a instrução. O local em que o cursor se encontra possui a cor do fundo totalmente preta, possibilitando a inversão da cor do caractere exibido, caso seja da mesma também.

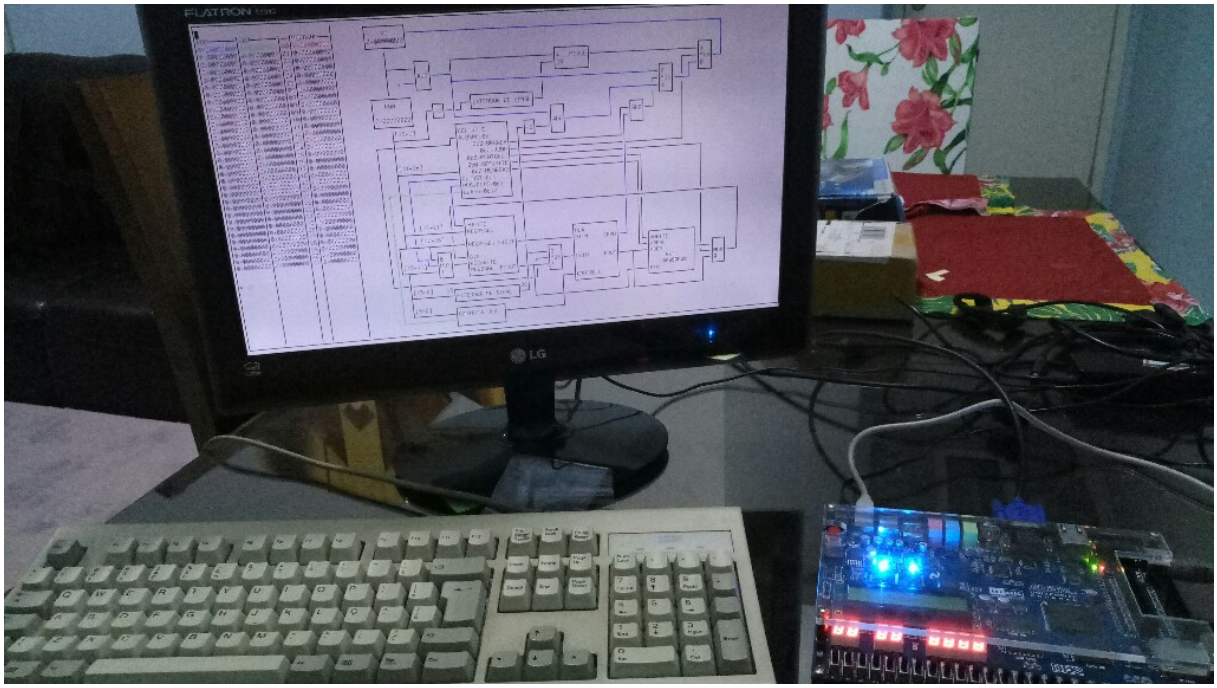
As linhas de dados possuem as seguintes cores para os seguintes valores que estiverem sendo enviados, por meio delas, os 3 bits menos significativos:

- Preto: 0;
- Azul: 1;
- Verde: 2;
- Ciano: 3;
- Vermelho: 4;
- Magenta: 5;
- Amarelo: 6 e 7;

A repetição da cor amarela para valores 6 e 7 se deve ao fato de a cor branca ser a mesma do fundo, então não seria possível visualizar.

Na figura 40, encontram-se o que é o kit da ferramenta (kit DE2), um teclado interface PS/2 e um monitor com interface VGA de resolução 1360x768.

Figura 40 – Ferramenta finalizada.



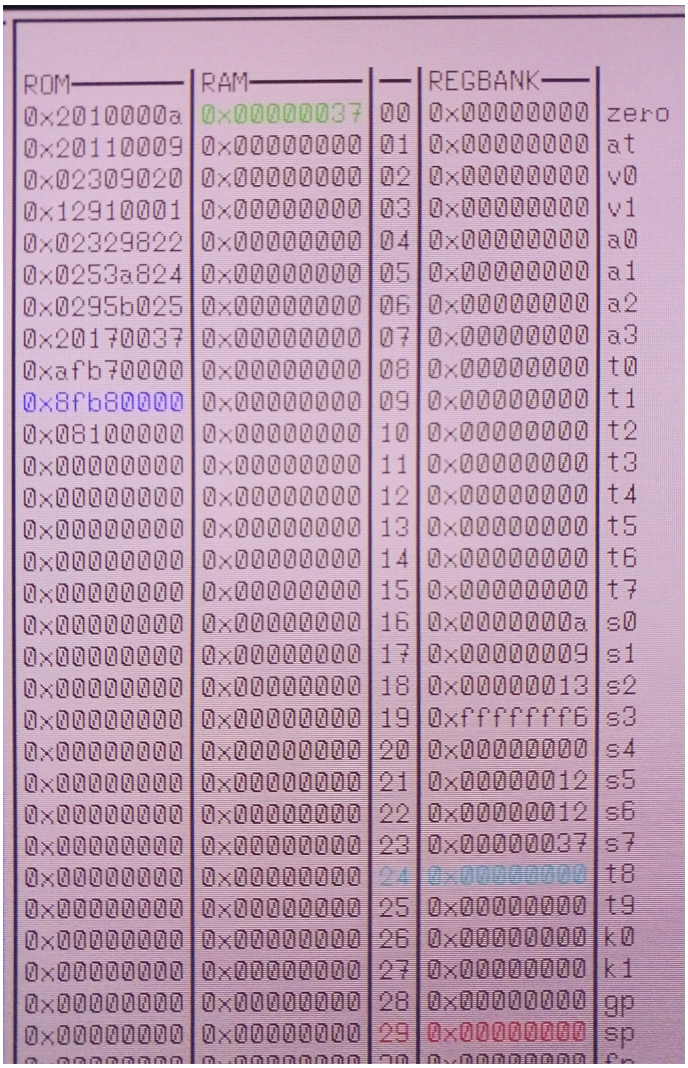
(Fonte: O Autor)

5.3 Analise de Execução de Instruções

Ao finalizar a escrita de instruções e armazená-las na memória, pode-se testar a execução das instruções e visualizar os resultados obtidos com o programa escrito. Na figura 41,

pode-se observar um programa escrito, com o propósito de testar a arquitetura MIPS monociclo.

Figura 41 – Exibição do programa gravado na memória ROM, dados na memória RAM e banco de registradores.



(Fonte: O Autor)

O programa escrito é o seguinte:

```
0   inicio:
1   ADDI $s0, $zero, 10
2   ADDI $s1, $zero, 9
3   ADD $s2, $s1, $s0
4   BEQ $s4, $s1, Branch
5   SUB $s3, $s1, $s2
6   Branch:
7   AND $s5, $s2, $s3
8   OR  $s6, $s4, $s5
9   ADDI $s7, $zero, 55
10  SW  $s7, 0($sp)
11  LW  $t8, 0($sp)
```

```
12 J inicio
```

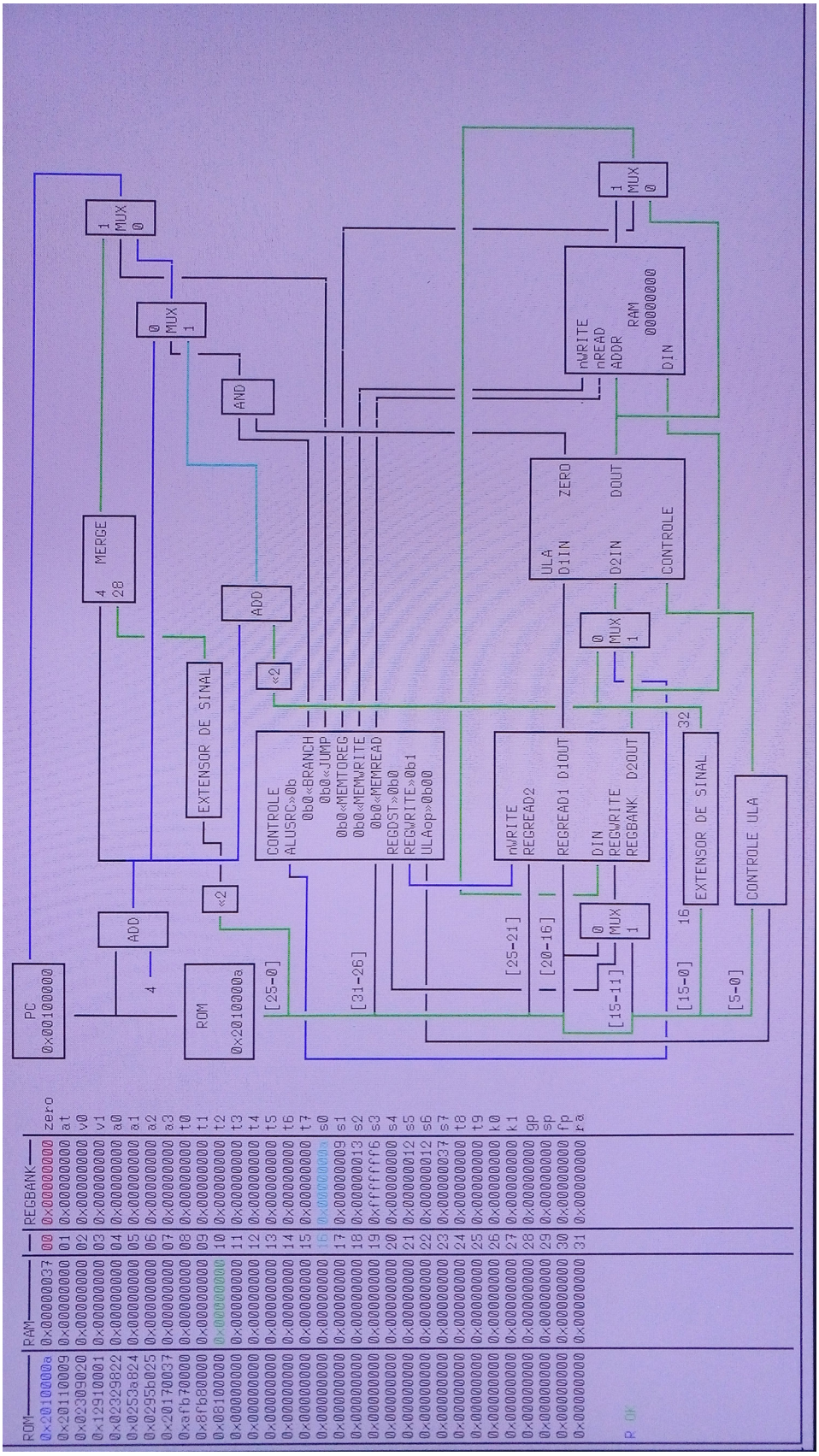
Analisando a figura 41 e o código, é possível analisar os resultados obtidos:

- As instruções ADDI, neste código, servem para iniciar as posições S0 e S1 do banco de registradores. Na figura, está exibido o valor 0x0000000a e 0x00000009 nas posições 16 e 17, respectivamente.
- A instrução ADD realiza a soma dos registradores S1 e S0 e armazena o resultado em S2, resultado em 13 da forma correta.
- A instrução BEQ verifica os valores S4 e S1, se verdadeiros, salta para o AND, caso contrário, executa o SUB. Como S4 e S1 são diferentes, ele passa a executar o SUB, o que demonstra que executou corretamente. Na figura 42, é possível visualizar quando a comparação é realizada, a saída $zero$ da ALU é mantida no valor 0, visto que a subs-tração (operação utilizada para comparar a igualdade dos valores) resulta em um valor diferente. Subtraindo S1 de S4 resulta em 0xfffffff7, o que não ativa a saída $zero$ e o fluxo de dados não permite a passagem da posição do desvio.
- Ao executar o SUB ele realiza a subtração de S2 em S1, resultando em 0xfffffff6, o que revela estar correta a execução.
- A instrução AND realiza a operação bit a bit dos registradores S2 e S3, resultando em 0x00000012, o que revela estar correto.
- A operação OR ocorre da mesma forma que o AND. Utilizando os registradores S4 e S5, resultando em 0x00000012, pois o registrador S4 possui o valor '0';
- É realizado o ADDI novamente para testar a inserção de instruções repetidas após um longo tempo, e também inserir o valor 0x00000037 no registrador S7. A execução desta instrução é verificada na figura 43 onde é possível observar que o caminho *REGWRITE-nWRITE* é desativado, permitindo a gravação no registrador e o valor é pego através da extensão de sinal;
- Então é armazenado na memória o valor presente em S7 com a instrução SW, na posição SP, com deslocamento de '0' posição.
- Agora é feita a leitura do dado armazenado e guardado em S8, algo que não ocorreu no momento da execução, o que demonstra que ele não puxa os dados antes da execução do pulso de clock.
- É inserido um Jump de forma a fazer um *loop* infinito e executar continuamente.

Nas figuras 42 e 43, é exibido, respectivamente, o caminho de dados com a execução das instruções BEQ e ADDI. Verifica-se que, quando a execução é focado no salto de posição de memória, o fluxo de dados é maior em todo o desenho, visto que é utilizada maior quantidade de recursos da arquitetura.Por sua vez, quando a instrução envolve os registradores e

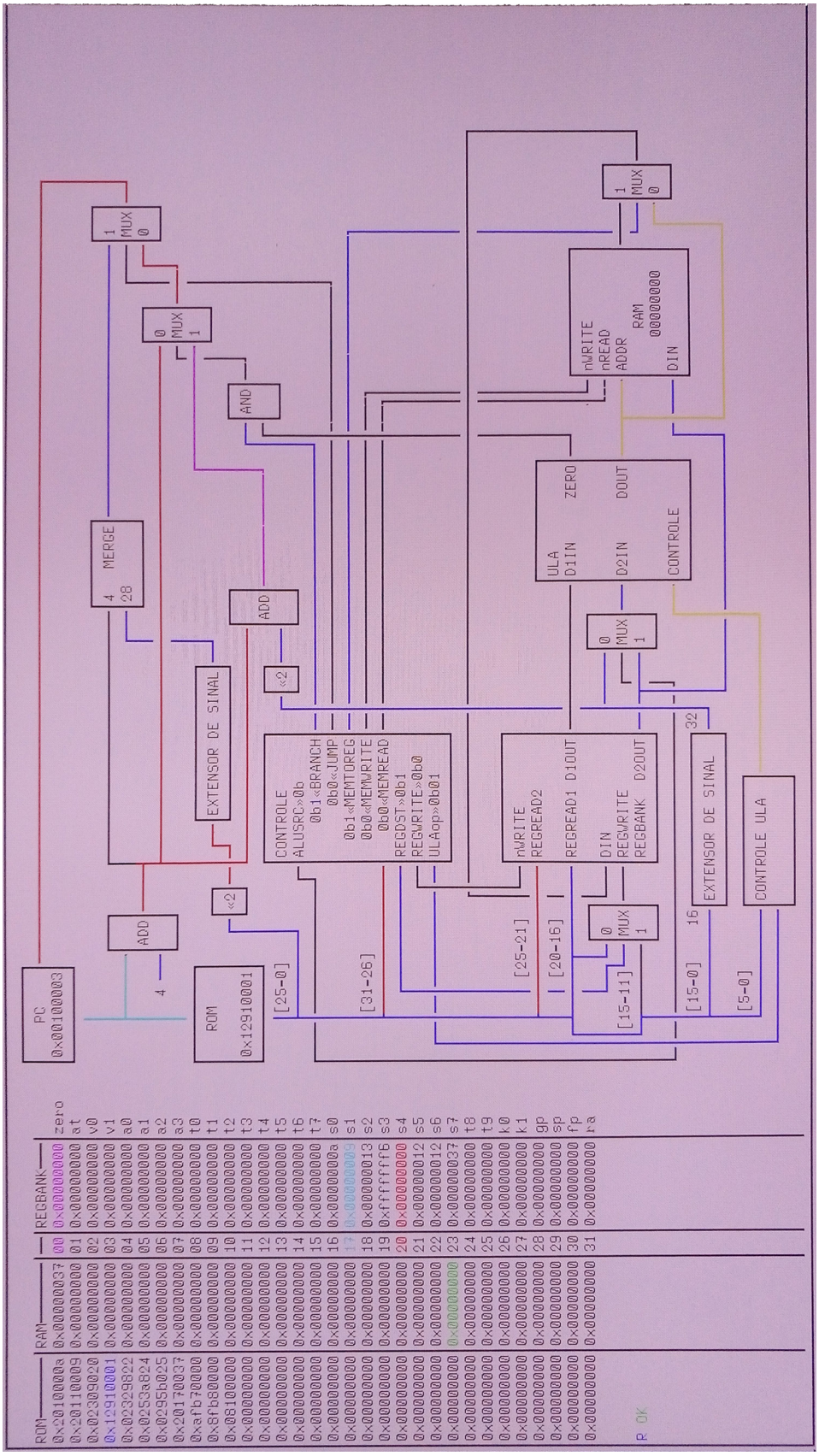
a memória RAM, o fluxo é maior na parte inferior. Essa análise pode ser feita observando a quantidade de linhas pretas (que representa o valor ’0’), no caminho de dados.

Figura 42 – Exibição da execução da instrução ADDI.



(Fonte: O Autor)

Figura 43 – Exibição da execução da instrução BEQ.



(Fonte: O Autor)

6 Conclusão e Trabalhos Futuros

“Tudo o que não nos destrói,
torna-nos mais fortes.”
Nietzsche

Este trabalho apresentou como objetivo projetar uma ferramenta que permitisse executar algumas instruções do MIPS, em seu modo monociclo prototipado em FPGA, com recursos visuais e interativos, os quais permitiriam exibir o caminho de dados e também programar para a arquitetura.

Um dos grandes desafios na implementação deste projeto foi encontrar uma maneira de exibir a imagem da arquitetura Mips. Embora as demais partes do projeto tenham se ocupado da interligação entre os blocos como um todo, exibir a imagem da arquitetura foi desafiante e imprescindível para o êxito da pesquisa em atingir o objetivo geral. Portanto, para desenhar a imagem da arquitetura MIPS no monitor foi implementado um software com o nome BlockPaint (nomeado pelo autor) que facilita fazer e/ou refazer o desenho da arquitetura por blocos de caracteres.

Desta forma, este programa possibilitou reduzir o tempo de desenvolvimento para produzir uma interface intuitiva, através da produção de um programa VHDL, ou seja, o BlockPaint permite criar automaticamente o programa VHDL da maneira como o usuário deseja como interface - imagem da arquitetura. Com isso, outras tecnologias disponibilizadas por kits de desenvolvimento FPGA podem ser usadas, permitindo ao programador maior facilidade na construção de uma interface, em particular com uso de um monitor VGA específico - isso é possível com adaptação simples no projeto por meio do software utilizado.

Importante destacar, como resultado, que o ambiente implementado produz uma interface com informações que permitem visualizar o caminho de dados da arquitetura MIPS, a qual altera as cores dos caminhos, conforme instrução executada, facilitando assim a compreensão da visualização da execução das posições selecionadas nas memórias de instruções, dos dados e do banco de registradores. A interface foi capaz de detectar os comandos enviados pelo usuário corretamente e o processo de montagem, feito de maneira instantânea, permitiu executar o código de imediato, assim como a sequência do programa inserido. No mais, novos códigos/programa podem ser criados, pois a arquitetura implementada é independente, ou seja, não há necessidade de embarcar o novo programa.

Quanto aos programas, estes, ao serem sintetizados, utilizaram uma grande quantidade de blocos lógicos (cerca de 10.000), o que impede o embarque de modelos de FPGA com quantidade inferior. Assim, a priori, a escolha da capacidade do dispositivo FPGA é relevante, pois ela é determinante para que a grande quantidade de processos desenvolvidos possa ser sintetizada com êxito. O objetivo proposto no presente trabalho foi alcançado, uma vez que foi implementada, com êxito, a interface que proporcionou a visualização e interação

do usuário com a arquitetura MIPS embarcada.

Quando se faz um paralelo da presente pesquisa com os trabalhos apresentados no capítulo 3, é possível identificar alguns avanços em relação. No trabalho de Llorente, Pulido e Rubio (2000) é feita uma implementação semelhante, mas, analisando as entrelinhas, a arquitetura MIPS implementada utiliza uma arquitetura reduzida, o que diminui a quantidade de instruções e não permite certa fidedignidade à arquitetura desenvolvida por Patterson e Hennessy (2012). Além dessa diferença na arquitetura, toda a execução é realizada inteiramente em um simulador de VHDL, ou seja, o MIPS não é prototipado em FPGA. Para que ele conseguisse implementar tais recursos, teria que produzir os mesmos controladores implementados na presente pesquisa.

No trabalho MIPSFPGA, toda a arquitetura implementada segue todos os detalhes da arquitetura explicada no livro de Patterson e Hennessy (2012). O recurso para gerar o código em Verilog, respectivo da arquitetura, permite a prototipação do projeto em FPGA, porém há limitações como o uso de apenas um programa por sintetização, além da visualização e interação com a arquitetura estarem limitada aos recursos presentes no kit DE2 que foi utilizado. Comparando com a implementação realizada nesta pesquisa, é possível visualizar a simplificação com relação à interação, utilizando um teclado, não se limitando a *switches* e botões do kit, além de permitir visualizar a execução em um monitor e programar para a arquitetura, sem precisar realizar a sintetização novamente.

Por fim, para o autor, a presente pesquisa auxiliou na compreensão de produção de hardwares complexos, assim como proporcionou a experiência de utilizar ferramentas profissionais para a implementação do ambiente proposto.

6.1 Trabalhos Futuros

Como trabalhos futuros, são dadas algumas sugestões de modo a ampliar as funcionalidades e também avaliar o seu uso em alguns ambientes.

A memória utilizada é limitada aos blocos de memória internos do chip FPGA, a implementação de um controlador de memória externa que permita ampliar o armazenamento de dados, permite a redução de uso de blocos lógicos para o controle da mesma. Como a memória externa possui latência muito maior que a do chip, a utilização dos blocos internos como cache permite melhor desempenho em conjunto com o controlador.

A quantidade de blocos lógicos utilizados não permitiu a utilização da ferramenta completa no chip Cyclone IV, Como proposta,a ideia é buscar outros meios de implementação dos componentes, otimizando a lógica combinacional. Isso não se limita a apenas este chip, pois há uma gama de dispositivos com variados recursos, o que inclui uma grande variedade de blocos lógicos, dentre eles.

Esta implementação é limitada a apenas uma resolução, o que não é ideal. Apesar da tendência dos monitores em aumentar cada vez mais a densidade de pixels, é interessante adaptar um controlador que consiga detectar a resolução do monitor e ajustar a sua imagem. Isso posto, o podem ser levantados os seguintes questionamentos: seria possível uma forma

de detectar a resolução e se adaptar automaticamente? A inserção manual de códigos de outras características de monitores é suficiente e menos dispendiosa como tempo de trabalho?

Como as tecnologias de interfaceamento utilizadas são antigas (PS/2 e VGA), apesar de estarem consolidadas, podem cair em desuso com o avanço tecnológico. Com isso, a implementação de novos controladores de interface para dispositivos recentes, como o HDMI (High-Definition Multimedia Interface) ou USB (Universal Serial Bus), permitirá o aumento do conjunto de dispositivos compatíveis.

Como os dispositivos FPGA estão evoluindo e tendo seus preços reduzidos, muitas instituições de ensino estão adquirindo kits para o ensino da Arquitetura de Computadores. Como foi informado no capítulo 1 desta pesquisa, muitos trabalhos são desenvolvidos utilizando kits. Destes trabalhos, muitos possuem avaliação de uso em sala de aula, e como possuem limitação visual e interativa podem desestimular o seu uso.

Com isso, o ambiente desenvolvido nesta pesquisa pode ser utilizado em aulas de Arquitetura de Computadores e, como trabalho futuro, pode-se avaliar o uso desta ferramenta em relação a outras, de forma a comprovar sua real eficácia no ensino.

Referências

ALBERTINI, B. C. A vez do fpga. *O Setor Elétrico*, O Setor Elétrico, v. 110, p. 184–185, mar. 2015. Disponível em: <http://www.ieee.org.br/wp-content/uploads/2014/05/Ed110_EspacoIEEE.pdf>. Citado nas páginas 17, 18 e 22.

BONATO, A. C. *PROJETO DE UMA INTERFACE DE MEMÓRIA DDR*. 2008. 64 p. Tese (Monografia) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, Porto Alegre, Rio Grande do Sul, BR, 2008. Disponível em: <http://www.lapsi.eletro.ufrgs.br/producao/outros/Relatorios_tecnicos/timemoriaddr.pdf>. Citado nas páginas 33 e 35.

CHAPWESKE, A. The ps/2 mouse/keyboard protocol. 2003. Disponível em: <http://antoni.sterna.staff.iiar.pwr.wroc.pl/ucsw/PS2_keyboard.pdf>. Citado na página 27.

ENGEL, T.; SILVEIRA, D. *Métodos de Pesquisa*. Editora UFRGS, 2009. Disponível em: <http://www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf>. Citado na página 46.

FARIAS, G.; MEDEIROS, E. S. *Introdução à Computação*. João Pessoa, PB: Editora da UFPB, 2013. Citado na página 37.

FONSECA, J. *Metodologia da pesquisa científica*. Fortaleza, CE: LTC - Livros Técnicos e Científicos Editora S.A., 2002. Citado na página 46.

HAUCK, S.; DEHON, A. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Elsevier Science, 2010. (Systems on Silicon). ISBN 9780080556017. Disponível em: <https://books.google.com.br/books?id=dYKmZy0asrsC>. Citado na página 21.

HENNESSY, J.; PATTERSON, D. *Arquitetura de Computadores: Uma Abordagem Quantitativa*. CAMPUS - RJ, 2008. ISBN 9788535223552. Disponível em: <https://books.google.com.br/books?id=5LjyQgAACAAJ>. Citado nas páginas 15, 29, 30, 31, 32, 40 e 51.

HSIEH, G.; HUSH, J. C. Phase-locked loop techniques - a survey. *IEEE Transaction on Industrial Eletronics*, IEEE, v. 43, n. 6, p. 609–615, dez. 1996. ISSN 0097-8418. Disponível em: <http://www.etc.tuiasi.ro/cin/Downloads/pll/PLL_Survey.pdf>. Citado na página 22.

HYNIX. *SDRAM Device Operation*. 2003. Acessado: 2018-12-28. Citado na página 36.

HYNIX. *Datasheet SDR SDRAM Hynix HY57V641620FTP*. 2007. Acessado: 2018-11-15. Citado nas páginas 34, 35, 36 e 69.

KABIR, M. T.; BARI, M. T.; HAQUE, A. L. Visimips: Visual simulator of mips32 pipelined processor. In: *2011 6th International Conference on Computer Science Education (ICCSE)*. [S.l.: s.n.], 2011. p. 788–793. Citado nas páginas 15, 16 e 17.

LI, G. H.; WU, Z. Design and realization of sdram controller based on fpga. In: *Industrial Instrumentation and Control Systems*. [S.l.]: Trans Tech Publications, 2013. (Applied Mechanics and Materials, v. 241), p. 2233–2237. Citado na página 35.

LLORENTE, J. M. A.; PULIDO, N. P.; RUBIO, J. B. A visual simulation environment for mips based on vhdl. *Computers and Education in the 21 st Century*, v. 16, n. 8, p. 55–63, 2000. Citado nas páginas 38, 39, 40, 41 e 85.

MENEZES, P. B. *Linguagens Formais e Autômatos*. [S.l.]: Luzzato, 2000. Citado na página 22.

OLIVEIRA, A. F. S. *Sistema didáctico de baixo custo com FPGA de alta densidade*. Porto, Portugal: [s.n.], 2012. 164 p. Disponível em: <http://recipp.ipp.pt/handle/10400.22/4421>. Citado na página 20.

PATTERSON, D.; HENNESSY, J. *Computer Organization and Design: The Hardware/Software Interface*. Elsevier Science, 2012. (Morgan Kaufmann Series in Computer Graphics). ISBN 9780123747501. Disponível em: <https://books.google.com.br/books?id=DMxe9AI4-9gC>. Citado nas páginas 15, 29, 51, 72 e 85.

PENHA, J. C.; FONTES, G.; FERREIRA, R. Mipsfpga - um simulador mips incremental com validação em fpga. *International Journal of Computer Architecture Education (IJCAE)*, p. 19–25, 2016. Citado nas páginas 29, 39 e 41.

PINOUTS. *VGA pinout diagram*. 2019. Acessado: 2019-02-03. Citado na página 25.

POLIT, D.; BECK, C.; HUNGLER, B. *Fundamentos de pesquisa em enfermagem: métodos, avaliação e utilização*. Artmed, 2004. ISBN 9788573079845. Disponível em: <https://books.google.com.br/books?id=Nh2WAAAACAAJ>. Citado na página 46.

RUDRAMMA, K. R.; KRIHNA, B. M. Ps2 vga peripheral based arithmetic application using micro blaze processor. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), v. 2, p. 153–157, ago. 2013. Disponível em: <https://pdfs.semanticscholar.org/fe5a/520a3255e1a20f9877ee4e867847f61587d4.pdf>. Citado na página 28.

RZRD. 睿智*FPGA*开发板*ALTERA IV EP4CE*四代*NIOSII SOPC* 全部*IO*支持外扩. 2018. Acessado: 2018-10-07. Citado na página 44.

SAMSUNG. 2015. Acessado: 2019-01-30. Disponível em: <https://www.samsung.com/br/support/model/NP500R5H-XD1BR/>. Citado na página 42.

SIPSER, M. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013. ISBN 113318779X. Citado nas páginas 22, 23 e 24.

SKLIAROVA, I. Desenvolvimento de circuitos reconfiguráveis que interagem com um monitor vga. *REVISTA DO DETUA*, p. 19–25, 2005. Disponível em: <http://revistas.ua.pt/index.php/revdeti/article/view/2071/1943>. Citado nas páginas 24, 26, 54, 55 e 56.

SOUSA, B. J.; JúNIOR, J. J. L. D.; FORMIGA, A. A. *Introdução à Programação*. João Pessoa, PB: Editora da UFPB, 2014. Citado na página 38.

TERASIC. *Altera DE2 Board*. 2006. Acessado: 2018-01-20. Citado na página 45.

VASCONCELOS, L. *Hardware Total*. [S.l.]: Makron Books, 2002. Citado na página 33.

VESA, V. E. S. A. *Monitor Timing Standard*. 2007. Acessado: 2018-11-15. Citado nas páginas 26, 59 e 64.

VETRA, V. S. C. *PS/2 PC Keyboard Scan Sets Translation Table*. 2019. Acessado: 2019-01-15. Citado na página 28.

VOLLMAR, K.; SANDERSON, P. Mars: An education-oriented mips assembly language simulator. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 38, n. 1, p. 239–243, mar. 2006. ISSN 0097-8418. Disponível em: <http://doi.acm.org/10.1145/1124706.1121415>. Citado nas páginas 15 e 16.

WIKIPEDIA. Vga connector. 2019. Acessado: 2019-02-03. Citado na página 25.